KVASIR: EXPLORATIONS IN MACHINE LEARNING
BY SEEING

By

Daniel Werner

A Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
Major Subject: Computer Science

Approved:

Selmer Bringsjord
Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

December 2007
(For Graduation December 2007)
© Copyright 2007
by
Daniel Werner
All Rights Reserved
# CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vii</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 What is Learning by Seeing?</td>
<td>2</td>
</tr>
<tr>
<td>1.2 The Importance of Learning by Seeing</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Focusing the Problem</td>
<td>5</td>
</tr>
<tr>
<td>1.3.1 The Key Class of Questions</td>
<td>5</td>
</tr>
<tr>
<td>1.3.2 Specific Objectives With Respect to These Questions</td>
<td>7</td>
</tr>
<tr>
<td>2. Preliminaries</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Logic-Based AI</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Theorem Provers</td>
<td>11</td>
</tr>
<tr>
<td>2.2.1 Resolution and Paramodulation</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2 SNARK</td>
<td>13</td>
</tr>
<tr>
<td>2.2.2.1 Sorts</td>
<td>14</td>
</tr>
<tr>
<td>2.2.2.2 Constants</td>
<td>14</td>
</tr>
<tr>
<td>2.2.2.3 Functions</td>
<td>14</td>
</tr>
<tr>
<td>2.2.2.4 Relations</td>
<td>15</td>
</tr>
<tr>
<td>2.2.2.5 Assertions</td>
<td>15</td>
</tr>
<tr>
<td>2.2.2.6 Proofs</td>
<td>16</td>
</tr>
<tr>
<td>2.3 Computer Vision</td>
<td>17</td>
</tr>
<tr>
<td>2.3.1 Black Box Approach</td>
<td>18</td>
</tr>
<tr>
<td>2.3.2 Figure Recognition</td>
<td>19</td>
</tr>
<tr>
<td>2.3.3 Optical Character Recognition</td>
<td>20</td>
</tr>
<tr>
<td>2.4 Logically Controlled Natural Languages</td>
<td>20</td>
</tr>
<tr>
<td>2.4.1 ACE: Attempto Controlled English</td>
<td>20</td>
</tr>
<tr>
<td>2.4.2 CELT: Controlled English to Logic Translation</td>
<td>21</td>
</tr>
<tr>
<td>2.5 Foundation in Geometry</td>
<td>21</td>
</tr>
</tbody>
</table>
LIST OF TABLES

2.1 Congruent Angles Formed by Two Parallel Lines and a Transversal . . . 22
LIST OF FIGURES

1.1 Machine learning model .............................................. 3
1.2 TIMSS Sample Problem 1 ........................................... 6
1.3 TIMSS Sample Problem 2 ........................................... 6
1.4 TIMSS Sample Problem 3 ........................................... 7
1.5 TIMSS Sample Problem 4 ........................................... 8
2.1 Black Box Example .................................................. 19
2.2 Example Transversal of Parallel Lines ......................... 22
2.3 Example of Angles formed by Intersecting Lines ............. 23
3.1 Example Problem Solved by Kvasir ............................. 24
3.2 Flowchart of Kvasir ................................................ 25
3.3 Line Simplification Algorithm by Douglas and Peucker, step 1 26
3.4 Line Simplification Algorithm by Douglas and Peucker, step 2 26
3.5 Line Simplification Algorithm by Douglas and Peucker, step 3 27
3.6 Line Simplification Algorithm by Douglas and Peucker, step 4 27
3.7 Line Simplification Algorithm by Douglas and Peucker, Final Step 27
3.8 Template for a Hetch .............................................. 31
3.9 Slate’s Reading Process .......................................... 35
A.1 Hetch Naming Scheme ........................................... 48
A.2 Glossary Example of Parallel Lines and a Transversal .... 49
A.3 Glossary Example of Intersecting Lines ...................... 50
ABSTRACT

This thesis presents a concept called “Learning by Seeing,” a new type of machine learning implemented in the system Kvasir, but with an emphasis on “reading” visual information. Designed with a focus on geometry problems for 8th grade math students, Kvasir reads both the visual and textual part of the problem, outputs the meaning a student would gather from viewing the question, finds the correct answer to the question, and justifies it. Kvasir begins to explore the idea that an Internet search engine like Google can answer questions from visual information as well as textual by focusing on this problem type. With much more work in “Learning by Seeing,” this will be possible.
CHAPTER 1
Introduction

For many years, it has been a goal in the field of computer science, specifically in Artificial Intelligence (AI), to have a system capable of answering a question posed by a human. To clarify, this question is asked using a natural language — one example could be “Who was the first President of the United States of America?” Most would simply type this into a search engine on the Internet such as Google or Yahoo!. The engine would attempt to return the website that best matches this query but would only accomplish it by doing a simple keyword search, possibly augmented with some “shallow” techniques. A related example is a test proposed by Alan Turing in 1950 now called the Turing test. In it, a human judge engages in a conversation with a human and a machine but cannot see either of them. If the judge cannot determine which is which, the machine passes the test. Here, if a machine existed that passed the Turing test, the machine would certainly return “George Washington” and not a list of website results.

Until recently, this goal of question answering (QA) has been impossible to realize in any form. It is still very limited. This is mostly because it is extremely difficult to determine the meaning of a sentence in English or any natural language. Additionally, questions can be asked about information that is in a non-textual form such as a picture. These questions are easily answered by humans although sometimes some background knowledge is required. For example, computers can answer very simple questions, such as: “Is there blue in this picture?” Computers can also do things that humans cannot do without assistance, like counting the number of pixels in a picture. Yet a computer is not sophisticated enough to understand a picture in the same way as a human and cannot grasp the overall idea or meaning.

At RPI in the Rensselaer AI and Reasoning Lab (RAIR Lab), however, systems have been developed with this problem in mind and significant gains have been made. Solomon is a new QA proof-of-concept that uses a form of controlled English called Attempto Controlled English (ACE). When given input in the proper form,
ACE can convert it to first-order logic. Then, using theorem provers integrated in Slate, another tool developed in the RAIR Lab, those controlled English sentences that have been converted to logic can be reasoned over and answers can be found. Even more significant is the fact that Solomon also answers questions pertaining to visual data using a combination of this same method, computer vision and other techniques. This process, which we in the RAIR Lab termed “Learning by Seeing” and implemented in the system Kvasir, will be the focus of this Master’s thesis.

1.1 What is Learning by Seeing?

“Learning by Seeing” is a new process in which visual and textual data is “read” by a computer, and represented as new knowledge, such that when the computer is asked a question, it can respond with the correct answer. As mentioned before, this process does involve computer vision, but is not simply the processing of images; it is something much larger, since it involves robust declarative knowledge. Vinay Shet follows a similar approach to develop a visual surveillance system that monitors the activity of people and detects violations of rules that are allowed by the people, such as unauthorized entry of buildings. VidMAP, his system, uses low-level computer vision techniques to log events and higher-level reasoning techniques to determine whether the events are violations.

Historically, computer vision is a well-developed field of computer science in which techniques already exist to recognize objects and shapes, extract text from images, and interact with human users — among many others. Yet, none of these have a form of intelligence about the work; rather they perform simple routines. No work will be done here to delve further into advancing research on computer vision, but this thesis will explore how a computer can take knowledge gained from processing images, analyze it for proper significance, convert it into logic, and then answer questions from a user on the examined image. This process, when viewed as a whole, is a kind of machine learning, a field of AI.

Traditionally, there are three types of machine learning: supervised, unsupervised, and reinforcement learning. “Learning by Seeing” explores a new method of learning that is so far ignored. The other types of learning have multiple trials
that usually involve a training set of data or a given reward if the answer is correct. An example schematic from Russell and Norvig’s book [25] can be seen in Figure 1.1. In it, there are other components of learning, such as sensors and actuators, which allow for interaction with the environment, but the important segment is the cycle of information that flows through the learning element, the problem and the performance or evaluation part. Instead, “Learning by Seeing” attempts to learn the correct answer in a single trial. At the beginning of the trial, it must have all the information that it needs to solve the problem. In contrast, the other methods develop their function to solve a problem, which often is a probabilistic approach, as the training data is processed. Once enough trials have been completed, the machine typically does not change the decision function.

To further describe the difference between “Learning by Seeing” and computer vision, one can compare it to the semantic difference between hearing and listening. In hearing, a person recognizes that another is talking. Listening requires one to not only hear but to pay attention to the actual words that are coming from the other’s mouth, determine the meaning of what they are saying, and then decide

![Figure 1.1: Machine learning model](image-url)
a course of action based upon it. Here, computer vision is analogous to hearing while “Learning by Seeing” is analogous to listening. Computer vision looks at the images it processes and extracts some sort of data from it, but a system that analyzes the image to gather its overall meaning, and optionally uses that meaning for a purpose, can be said to have the power of “Learning by Seeing.” Basically, “Learning by Seeing” attempts to examine an image by the use of computer vision techniques but then takes what it has gathered from the examination and uses it to answer a question — which is the test of whether or not someone has learned.

The difference between Kvasir and Shet’s VidMAP is that his system does not “understand” the full meaning of what is happening. VidMAP logs events and then checks the events against a series of rules to see if any of the events are in violation of the rules — which is computer vision combined with some analysis of the data gathered. In contrast, Kvasir has a full grasp of the meaning of the image being examined and can answer full, valid questions about the image.

Unfortunately, the test of learning mentioned above, and the phrase “Learning by Seeing,” still remains undefined, unclear, and vague to the reader as to what it might look like or how it can be implemented. Ultimately, while it has an anthropomorphic name, it boils down to programming a computer to understand and overcome the questions that are posed to it. Greater detail on the specific questions chosen follows in Section 1.3 and an in-depth study of Kvasir can be found in Chapter 3.

1.2 The Importance of Learning by Seeing

One can question why it is important for a machine to be able to learn by reading visual data. The explanation has already been touched on. In the same way that we cannot ask a question in a Internet search engine and receive an answer that was determined from the meaning of the question, we certainly cannot ask questions about images. Questions such as “How many people are in the picture?” or “What type of scene is this?” are examples of questions a person might ask of a specific picture. A query on a search engine such as “What building is Martin Luther King Jr. standing in front of during his ‘I have a dream’ speech?” is something
that a person might ask when viewing pictures from the event, and, if Google could read images, it could answer the question by just looking at images. While this question likely could be answered from just reading textual documents using the normal keyword search strategy, one simple answer could not be returned. Instead, numerous webpages would be returned which could possibly contain the answer.

Internet search engines are only one example of a use for this technology. They are likely the most interesting as they search over huge amounts of data stored in various forms with millions of queries daily. Yet, presumably, anyone can see that this technology would be extremely useful in many different fields.

### 1.3 Focusing the Problem

The questions posed will be directed towards images that are limited to a specific type of question. Without focusing the problem, attempting to do this analysis would be impossible because of the immensity of the task. The concentration will instead be on solving the questions created by the Trends in International Mathematics and Science Study (TIMSS)\(^{[16]}\); specifically those given to 8th grade math students. These questions range from numerical questions about fractions and decimals, to ratios and proportions, and ones involving algebra, relationships, and patterns. They relate specifically to this thesis because the images that are part of the questions have additional but necessary information to the answering of the question.

#### 1.3.1 The Key Class of Questions

Of those mathematical types of questions, geometry questions will be the focus of this thesis. The topics can be on lines and angles, congruence, symmetry, and circles, just to name a few. The process needed to answer the questions can range from reasoning, to just solving routine problems using memorized facts directly. Some example problems taken from the TIMSS released set of problems for eighth grade mathematics in 2003\(^{[2]}\) are shown below.

Figure 1.2 shows an example where a student needs to apply concepts learned about congruence and similarity, while Figure 1.3 is a simpler question that is solved
In square $EFGH$, which of these is FALSE?

A. $\triangle{EIF}$ and $\triangle{EIH}$ are congruent.
B. $\triangle{GHI}$ and $\triangle{GHF}$ are congruent.
C. $\triangle{EFH}$ and $\triangle{EGH}$ are congruent.
D. $\triangle{EIF}$ and $\triangle{GIH}$ are congruent.

Figure 1.2: TIMSS sample problem involving congruence and similarity

The figure above is a regular hexagon. What is the value of $x$?

Answer: ____________

Figure 1.3: TIMSS sample problem on regular polygons

using knowledge about shapes — that regular polygons have certain properties about the measure of the sum of the interior angles. A problem that involves reasoning is more difficult because it requires that a student already knows the facts and procedures, can apply the concepts, and then can combine this information to solve the problem, thinking strategically about how it all fits together. There is no simple formula to input numbers into; the correct pieces of knowledge must be assembled. Figure 1.4 is an example of this.

However, the main geometric type of problem chosen can be seen in Figure 1.5. In this example there are two parallel lines intersected by a third line. There are inherent properties of this object, as long as the lines are parallel, which allow for many different types of questions. Specifically, the angles are of the the most interest. Each of the other problems mentioned can be solved by the process developed in
RPI’s RAIR Lab, yet we chose this type specifically for a few reasons. The properties in this type of figure allow for many different types of questions. Also, the figure is commonly seen on its own or combined with other figures. Other questions involve widely varying shapes that have similar properties, but since the shapes are so different, much more work must be done to allow for each type of question.

1.3.2 Specific Objectives With Respect to These Questions

The system Kvasir has several objectives that it attempts to accomplish on each image it analyzes:

Objective 1 Read the question and problem description and output the same understanding a student would have after viewing the image.

Objective 2 Select the correct answer from among the possible answers in the multiple choice question and provide a justification of it.

Reading the question, both visual data and text, is the difficult part of the process. Once the meaning of the image is understood, answering questions posed should be quite simple. In this problem set, only one of the possible answers should
Figure 1.5: TIMSS sample problem on parallel lines and an intersecting line

be correct and is called the key here — the other possible answers will be referred to as distractors. Simple as it may be, this test of returning the correct answer is important to prove that the interpretation is accurate.
CHAPTER 2
Preliminaries

In this chapter, background information will be presented on all of the topics that this thesis will cover. Terms, meanings, and technologies will be briefly defined so that the reader can later understand all that this thesis undertakes.

2.1 Logic-Based AI

The RAIR Lab[1] uses a logic-based approach to AI[6, 7, 24] which influences the systems it develops. There are many reasons why such an approach is appropriate. Simply, there is such a firm foundation in computer science on logic, with many AI books written using this method, that it makes it a wise choice to continue on in this path. Also, any proofs found can be clearly and mathematically justified using logic[3] — something that cannot be done without it. It is essential for Kvasir to have this ability because the type of problems it is answering need justification for their validity. Many other reasons exist as to why logic is very useful but that is not the purpose of this thesis. Instead, a short description of logic-based AI follows to prepare the reader for the subsequent description of Kvasir.

Nilsson (1991) gives three theses that logic-based AI rests on: that the machines have knowledge of their environments, that they represent much of their knowledge declaratively, and that the declarative knowledge can be used at least as well as first-order predicate calculus. Central to these formulas is the fact that all the declarative knowledge is stored in the form of sentences. This becomes the knowledge base from which the machine understands the world. The sentences consist of objects, relations, and functions combined using the predicate calculus to declare what knowledge of the world the machine should have. The sentences may be simple things like:

(color sky blue)

which simply gives the machine the knowledge that the color of the sky is blue. This is one method by which the machine gets information about the world.
As the machine interacts with the world it is placed in, the sentences can be changed and added to. This second method by which the machine gets new knowledge is from reasoning. For example, the machine might later learn that the sky is black at night. And then it might reason that sometimes the sky is blue, sometimes it is black, that during the day it is blue, and so on. Each time the sentences are changed, it can be said that the world is formalized in a new state of the machine. When new sentences are created based upon a computation on a set of sentences, they are said to be derived and are added to this knowledge base.

And so the knowledge base grows based upon methods of reasoning and interaction with the world. While this world may not be accurate, it is still called the machine’s knowledge and not its beliefs. It is the designer’s guesses and view of the world. Something that is both important and interesting is that even if the designer is wrong, the knowledge can be used for purposes other than what is intended because of the form of knowledge representation. For example, if a designer represented the students in a class as knowledge, including data on names, backgrounds, place of birth, majors, and grades, it could have been for the purpose of analyzing the correlation of grades and origins. However, the same data could be used by another machine that gathered statistics about spread of students in classes.

To give a syntactical background in logic, there are several items that need to be described. Each variable in the infinite alphabet of propositional logic can be connected via five operators: $\neg$, $\rightarrow$, $\leftrightarrow$, $\land$, and $\lor$. They are read as ‘not,’ ‘implies’ (or ‘if ... then ...’), ‘if and only if,’ ‘and,’ and ‘or,’ respectively. One could then say that ‘if the sky is blue, then it is daytime, and the sun is out’ by writing

$$p \rightarrow (q \land r)$$

with the variables $p$ as ‘the sky is blue,’ $q$ as ‘it is daytime,’ and $r$ as ‘the sun is out.’ The use of the quantifiers $\exists x$ (‘there exists at least one $x$ such that ...’) and $\forall x$ (‘for all $x$ ...’), along with the addition of the $=$ symbol, make propositional logic become first-order logic. The first is called the existential quantifier and the second the universal quantifier.

Issues do arise upon using this approach to AI. Conceptualizing the world as
objects, relations, and functions, as well as determining what to say, can be extremely difficult. The syntax of how it shall be written and the means of interaction are understood because of the use of logic and predicate calculus, but the challenge of putting what the world actually is into logic remains. Additionally, there is the issue of unsound inferences. For example, they may appear when the designer errs in inferring, typically based upon consistent data, that something is always true when in fact a case where it does not hold just has not yet been seen.

2.2 Theorem Provers

What naturally follows from a logic-based approach to AI is a means or method to take a formula and verify if it is provable or not by checking against the knowledge base. Theorem provers, of which many exist, are the implementations by which this is done. They differ in many ways, including whether or not they require human interaction or solve the problem on their own. To verify a theorem, or see if it can be proved from the knowledge base, they use the following property:

\[ \alpha \vdash \beta \text{ if and only if the sentence } (\alpha \land \neg \beta) \text{ is unsatisfiable} \]

This means that if \( \alpha \) is the knowledge base and \( \beta \) is the theorem, then the theorem can be proved if the conjunction of the knowledge base and the negation of the theorem is not true in any possible model (\( \alpha \vdash \beta \) means \( \alpha \) yields \( \beta \)).

This is a common way of finding the correct answer and is known as refutation, or originally, *reductio ad absurdium*. In this technique, the negation of the theorem that is being checked for validity is inserted into the knowledge base. Together this can be represented as \( \alpha \land \beta \), again, using the assumptions for what \( \alpha \) and \( \beta \) are from above. Then a check for consistency is run and if a contradiction is found, the theorem is true. If this proof-by-contradiction method does not find the correct answer, it can continue looking forever either because the theorem is wrong or the knowledge base does not contain enough information to determine the answer. To avoid this, a time limit is typically set, which when exceeded, can be taken as an indication that one of those two previously mentioned options exists.
2.2.1 Resolution and Paramodulation

Resolution and paramodulation are two means of inference used by a variety of theorem provers, including Otter\textsuperscript{[21]} and SNARK\textsuperscript{[29]}, that assist them in finding a refutation. Resolution allows the system to simplify the sentences in the knowledge base while paramodulation allows the substitution of equivalent clauses or atoms in sentences. Stickel, Waldinger and Chaudhri give examples of each in their \textit{A Guide to SNARK}\textsuperscript{[30]} while the definitions here are from the book \textit{Artificial Intelligence: A Modern Approach (2\textsuperscript{nd} Edition)}.

Before the rules of resolution and paramodulation can be given, several things have to be explained. Inference, a term mentioned above, has a specific way of being written that will show up in the descriptions of resolution and paramodulation. It is easiest to show the means of writing it with an example, specifically the rule of Modus Ponens:

\[
\frac{\alpha \rightarrow \beta, \alpha}{\beta}
\]

This notation means that if a knowledge base contains both the sentences $\alpha \rightarrow \beta$ and $\alpha$ then the sentence $\beta$ can be inferred. A substitution for a variable $x$ with a constant $\text{John}$ can be written $\{x/\text{John}\}$. In a rule of inference this appears as $\text{SUBST}(\theta, \alpha)$. Lastly, unification occurs when different logical expressions can be made to look identical by using substitution. It is written:

\[
\text{UNIFY}(p, q) = \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)
\]

and a unifier is returned if one exists. For example:

\[
\text{UNIFY}(\text{Knows(John,x)}, \text{Knows(John,Jane)}) = \{x/\text{Jane}\}
\]

is an example where $\{x/\text{Jane}\} = \theta$ and is a valid substitution returned by the unification process\textsuperscript{[25]}.

At this point, the basics have been presented and resolution and paramodulation can be explained. Resolution is an inference rule that assists the refutation proof technique. Russell and Norvig state that two clauses can be resolved if they share no variables and they contain complementary literals. Complementary literals are literals that are the negation of each other. In the following rule, let $l_i$ and $m_j$ be literals for $1 \leq i \leq k$ and $1 \leq j \leq n$, respectively, and let $l_a$ and $m_b$ be
complementary literals. The resolution rule then follows:

\[
\frac{l_1 \lor \cdots \lor l_k, \ m_1 \lor \cdots \lor m_n}{l_1 \lor \cdots \lor l_{a-1} \lor l_{a+1} \lor \cdots \lor l_k \lor m_1 \lor \cdots \lor m_{b-1} \lor m_{b+1} \lor \cdots \lor m_n}
\]

When applying resolution, if an identical sentence can be created via substitution, then it will be inserted into the knowledge base. Here, (2.1) and (2.2) give the sentence (2.3) using resolution because both \( P \) and \( \neg P \) cannot be true.

\[
\begin{align*}
(\text{or } (\text{not } P) \ Q) & \quad (2.1) \\
(\text{or } P \ R) & \quad (2.2) \\
(\text{or } Q \ R) & \quad (2.3)
\end{align*}
\]

Paramodulation can be described in a less technical example using relations. If a user were to declare a rule that ‘if someone is a mother, then one is a parent,’ and the knowledge base contained the sentence that the mother of Carol is Betty, then paramodulation would obtain the sentence that Betty is a parent of Carol. To formally describe this, for any terms \( x, y, \) and \( z \), where \( \text{UNIFY}(x, z) = \theta \):

\[
\frac{l_1 \lor \cdots \lor l_k \lor x = y, \ m_1 \lor \cdots \lor m_n[z]}{\text{SUBST}(\theta, l_1 \lor \cdots \lor l_k \lor m_1 \lor \cdots \lor m_n[y])}
\]

Basically, paramodulation is used to handle clauses where an equality literal appears. In the following example, (2.6) follows from (2.4) and (2.5) as long as \( P[s] \) contains at least one occurrence of \( s \).

\[
\begin{align*}
(\text{or } P[s]) & \quad (2.4) \\
(\text{or } (= \ s \ t) \ Q) & \quad (2.5) \\
(\text{or } P[t] \ Q) & \quad (2.6)
\end{align*}
\]

2.2.2 SNARK

SNARK, or SRI’s New Automated Reasoning Kit, is the theorem prover that is used in conjunction with Kvasir and so a description of it is important for the reader. As mentioned above, SNARK uses refutation to find a proof but also allows
the human user to explicitly select that either resolution or paramodulation or both be used to assist the refutation. Additionally, it is an automated theorem prover like Otter, which means that no human interaction is required during run-time and is built for use with multi-sorted first-order logic. Lastly, it is implemented in a subset of Common Lisp and so the knowledge base and all theorems also are entered in Lisp form. Further descriptions of the basics of SNARK follow, and while many of the examples are taken from *A Guide to Snark*, that guide is out of date and therefore some of the syntax has been updated.

2.2.2.1 Sorts

A sort is similar to a type or object. For example, if a user were dealing with relations between people, the user would wish to declare a *person* sort as seen below.

```
(declare-sort 'person)
```

Then if the user were adding the relations of parent and mother that were mentioned in the example in Section 2.2 he would declare both Carol and Betty as of the sort *person*. The fact that SNARK uses multi-sorted first order logic means that there can be many different sorts.

2.2.2.2 Constants

All constants used in SNARK must be declared first. This consists of a simple `declare-constant` followed by the name of the constant and then the sort. Here is an example with Carol and Betty being declared as of the sort person.

```
(declare-constant 'Carol :sort 'person)
(declare-constant 'Betty :sort 'person)
```

2.2.2.3 Functions

Functions are important because they take a set of objects and return a sorted value. Often they are used to set a constant equal to the return value of a function but they are more powerful when used in conjunction with relations and assertions. This is the declaration of the function *mother* followed by an example of how to use it:
(declare-function 'mother 1 :sort '(woman person))
(assert '= (mother Carol) Betty))

Here, Betty is set to be equated with the value returned from the function mother that was called with the argument Carol. The word assert is simply a way of placing the sentence that Carol’s mother is Betty into the knowledge base. A more detailed description of assert can be seen below in Section 2.2.2.5.

2.2.2.4 Relations

Relations are a means of creating a link between two variables. If Dave were declared as a person, then one would read this as Dave is the spouse of Betty. The declaration of a relation precedes that example.

(declare-relation 'spouse 2 :sort '(person person))
(spouse Dave Betty)

2.2.2.5 Assertions

Assertions are useful in several ways. First, they can be a means of simply inserting sentences into the knowledge base. For example, assuming the parent relation is defined, one might wish to assert that Betty is the parent of Carol. This can be easily done by placing an assert around the use of the parent relation.

(assert '(parent Betty Carol))

Also, they can be used to insert rules into the knowledge base. One might wish to require that if Dave is the spouse of Betty then Dave cannot have any other spouses. Or assuming the brother relation is declared, one can assert that women are not brothers. Basically this says that if x is a woman, and y is a person, then x cannot be a brother of the person.

(assert
 ‘(implies
   (and (woman ?x) (person ?y))
   (not (brother ?x ?y)))
 :name ‘women-are-not-brothers)
Here a few other concepts are brought up. In SNARK, \texttt{implies} is the way to say \textit{if}. The question mark in front of both $x$ and $y$ means that they are universally quantified variables but don’t have to be declared as sorts. So SNARK will attempt to use any constant it finds declared in place of $x$ and $y$. Lastly, the :\texttt{name} followed by \texttt{‘women-are-not-brothers} is just a method of declaring a name for the relation. Later, when a proof is looked for, this name will be inserted whenever that assertion is used. If a name is not declared, a number is inserted instead. When reading the proof, which can be very long, having names instead of numbers for objects greatly simplifies the process.

\subsection*{2.2.2.6 Proofs}

Finally, after setting up SNARK to deal with different types of objects, inserting sentences into the knowledge base, declaring functions and relations, and then creating rules for how the sentences can interact with each other, one gets to the main goal of using SNARK: proving theorems. To use it is merely \texttt{prove} followed by the theorem in question. Upon execution, all the output from SNARK is displayed as it searches for the proof, and if an answer is found, a new section appears with the refutation of the negation of the theorem — as covered in Section 2.2. For example, if one wondered \textit{“Who is Carol’s grandparent?”}, one could formalize that in logic as the theorem that there exists a person who is a grandparent of Carol.

\begin{verbatim}
(prove ‘(grandparent ?z Carol) :answer ‘(ans ?z))
\end{verbatim}

The :\texttt{answer} ‘(ans ?z) component tells SNARK to return what value of $?z$ satisfies the question. Several possible answers exist, namely:

\begin{verbatim}
(father (father Carol))
(father (mother Carol))
(mother (father Carol))
(mother (mother Carol))
\end{verbatim}

The answer is returned in the refutation. The refutation for the first answer follows.

\begin{verbatim}
(Refutation
  (Row father-is-parent
    (parent (father ?x) ?x)
    assertion)
  (Row parent-of-parent-is-grandparent
    (parent (father (father ?x)) ?x)
    assertion)
\end{verbatim}
(or
  (not (parent ?x ?y))
  (not (parent ?y ?z))
  (grandparent ?x ?z))
assertion)
(Row does-carol-have-a-grandparent
  (not (grandparent ?x carol))
~conclusion
Answer (ans ?x))
(Row 12
  (or (not (parent ?x ?y)) (not (parent ?y carol)))
  (resolve
does-carol-have-a-grandparent
  parent-of-parent-is-grandparent)
Answer (ans ?x))
(Row 23
  (not (parent ?x carol))
  (resolve 12 father-is-parent)
Answer (ans (father ?x)))
(Row 30
  false
  (resolve 23 father-is-parent)
Answer (ans (father (father carol))))

The answer can be seen on the final line of the refutation. While they do not seem to be actual answers, since father and mother are functions, executing the answer would return the variable that a user would expect. One thing to note: SNARK will only return one answer at a time. Executing (closure) will ask SNARK to find a different proof for that theorem. It is not guaranteed that the answer will be different, but the proof will be. Yet eventually each of the four possible answers will be returned.

2.3 Computer Vision

Computer vision is a field in computer science that can be easily confused with image processing. Indeed, the meaning of the phrase “image processing” seems to be exactly what this thesis is covering. The scientific difference between computer vision and image processing is great, however.

Computer vision is the technology used to obtain information from images of many types, including video surveillance and medical imaging. Image processing
also works with images but instead requires that the images be manipulated, or processed, in some way to generate an output. The output may also be an image but does not have to be. Examples include image rotations, retouching and color corrections. There certainly is overlap between computer vision and image processing but there are definite distinctions as well. However, as this thesis covers an entirely new topic, it may be labeled as a sort of middle ground between the two. Because some define computer vision as “the science and technology of machines that see,” we will define the work done on Kvasir as a part of computer vision. And indeed, since a harder test called the total Turing Test that tests perceptual abilities requires computer vision\footnote{25}, the explorations demonstrated by Kvasir are important for the research needed to create a machine that can pass it.

Russell and Norvig state that computer vision is split into three types: low-level, mid-level, and high-level vision. Low-level vision consists of the smoothing of images to remove noise, and the extraction of parts of the image using techniques such as edge detection. The grouping of the edges into regions is considered mid-level vision, while lastly, in high-level vision, the regions are finally recognized as objects in the image’s world. As a whole, Kvasir can be considered falling into the high-level vision area. While parts of it fall into the other types of vision, this is to gather enough data, which requires using those lower level techniques, to be able to recognize the objects in the question.

Research in computer vision has been progressing for many years and can be extremely technical. This thesis is more concerned with the learning that is happening and so no work will be done to extend this topic. Instead a “Black Box Approach” will be used as explained in Section \ref{ss:2.3.1}.

\subsection{2.3.1 Black Box Approach}

Possibly the most common thing one might think of when hearing the phrase “black box” is a black box on an airplane that tells what went wrong in the case of an emergency. That is a correct use of the phrase but it is not limited to that implementation. Typically a black box is a system that has a series of inputs and outputs but what happens inside is disregarded or does not matter. In computer
science, this phrase is commonly used to refer to unseen or unknown algorithms and is the use implied here. An example is shown in Figure 2.1. One simply expects the black box to perform properly.

![Black Box Example](image)

**Figure 2.1: Black Box Example**

In this thesis, the black box is used when referring to the processing that is applied to the image. As mentioned above, no additional research is being done to extend computer vision. Since there are existing systems that can do the simple but necessary pre-processing required for Kvasir and they are free for use, they will be used with few, if any, modifications. So, the black box will take the image as input, do some processing and then output the basic information gathered. That output will be analyzed and will be the beginning of the real work done in “Learning by Seeing” — which is similar to the reasoning done by a student. Since what occurs inside the black box contributes to the final result, the technologies that are used will be covered to inform the reader. The work is divided into two parts: figure recognition in Section 2.3.2 and optical character recognition in 2.3.3.

### 2.3.2 Figure Recognition

Figure recognition often seems to imply that a computer is performing recognition human figures in an image[27]. Here, and in many other applications, the term means the detection of any kind of figure or object in the images. Typically, the method by which this is done falls into one of the following categories: learning what an object looks like by manually selecting it through a series of trials or by pre-defining what to search for.
2.3.3 Optical Character Recognition

Optical character recognition (OCR) is the phrase used to describe the conversion of certain images (ones with text in them) into text, using either a mechanical method or electronic processing. While OCR implies the use of some sort of optics which include mirrors and lenses, few programs actually use it anymore. Therefore the field of digital character recognition is encompassed as well by OCR. The most common application of OCR is in programs that are often bundled with a scanner. The programs take the images that were created from the scanned documents and perform character recognition on them and then output the text in a pdf or some text format[11].

2.4 Logically Controlled Natural Languages

Controlled natural languages are a subset of existing languages but have restrictions upon what is allowed, specifically in grammar and vocabulary. One can then split controlled natural languages into two parts: those that are designed to improve readability and those that assist in the processing of the text for meaning. Of these, logically controlled natural languages fall into the latter group[26].

Logically controlled natural languages are designed so that they can be mapped to first-order logic. To do this, they require the limitations on grammar. ACE (Section 2.4.1) and CELT (Section 2.4.2) are two examples of logically controlled natural languages, among several others.

2.4.1 ACE: Attempto Controlled English

One of the more mature controlled natural languages was created at the University of Zurich and is called Attempto Controlled English (ACE)[14, 15]. A subset of common English, ACE also has restrictions upon what can be used and what cannot be, but still allows a user to express what they wish. Indeed, when a human reads ACE text, they can easily understand it, but the restrictions in grammar and vocabulary make it such that without learning the rules of ACE, they cannot create it themselves.

The designers of ACE provide a parser called APE to translate English into
a variety of different outputs as long as it follows the rules set by ACE. It can be used both as a webclient and as a webservice, which is very useful in automating the process. Additionally, since the grammar defined by ACE is not all-inclusive, lexicons can be used to supplement ACE.

2.4.2 CELT: Controlled English to Logic Translation

Controlled English to Logic Translation (CELT) is another tool that makes use of controlled English. It is certainly less developed and polished than ACE, but what it attempts to do could make it much more powerful. ACE inspired Adam Pease and William Murray to develop CELT because one of the limitations of ACE is that it has only one meaning of a sentence. Instead, CELT’s vocabulary is thousands of words in size through the use of WordNet[13] and each is mapped to Suggested Upper Merged Ontology (SUMO)[23] concepts, which gives the option of multiple meanings. WordNet is a database of English words that are grouped by part of speech and are linked together both by concepts and meaning and by the lexical relation. SUMO is linked to all of the words in WordNet and has stored all of the possible meanings of each of the words there. Together, it provides CELT with a huge vocabulary whose words have multiple possible meanings, allowing CELT to have a broader understanding of what a user may be trying to say.

2.5 Foundation in Geometry

This thesis has focused the class of questions to be answered on the mathematical subject of geometry. Many of the terms and concepts that are fundamental to a geometry student may be new or long forgotten to a reader, so an overview will be given to cover or review those topics. In addition, a glossary of terms can be seen in Appendix A. Since it was important that the system here would work as a 8th grade math student would, Mathematics: Applications and Connections[9] was chosen to be a basis for how Kvasir was implemented. Many of the terms and definitions following have been taken from the book.

In Mathematics: Applications and Connections, the first thing a geometry student covers are parallel lines, or lines that in a plane that never intersect. The
Alternate Interior Angles
\[ \angle 2 \cong \angle 6 \]
\[ \angle 3 \cong \angle 7 \]

Alternate Exterior Angles
\[ \angle 1 \cong \angle 5 \]
\[ \angle 4 \cong \angle 8 \]

Corresponding Angles
\[ \angle 1 \cong \angle 3 \]
\[ \angle 2 \cong \angle 4 \]
\[ \angle 5 \cong \angle 7 \]
\[ \angle 6 \cong \angle 8 \]

| Table 2.1: Congruent Angles Formed by Two Parallel Lines and a Transversal |

The angles created by such a transversal of parallel lines have important qualities as they create congruent angles. **Congruent angles** are angles that have the same measure. When they are formed by parallel lines and a transversal, they have special names. **Alternate interior angles**, **alternate exterior angles**, and **corresponding angles** are named because of their locations in the figure. Table 2.1 lists the pairs of congruent angles (\( \cong \) is the symbol for congruency).

Moreover, angles formed by intersecting lines create congruent pairs. **Vertical
angles are opposite angles formed by the intersection of two lines and are congruent, while supplementary angles are two angles of which the sum of the measures is 180°. In Figure 2.3, \( \angle 1 \cong \angle 3 \) and \( \angle 2 \cong \angle 4 \) are vertical angle pairs, and \( \angle 1 \) and \( \angle 2 \), \( \angle 2 \) and \( \angle 3 \), \( \angle 3 \) and \( \angle 4 \), and \( \angle 4 \) and \( \angle 1 \) are supplementary angle pairs. One common way supplementary angles are formed is by these intersecting lines.

![Figure 2.3: Example of Angles formed by Intersecting Lines](image-url)
CHAPTER 3

Kvasir - Norse God of Inspiration and Knowledge

Here begins a thorough examination of Kvasir and the technology and algorithms that drive it. The analysis will be sequential, following the steps that are taken as a question is given to it. Figure 3.1 will be an example that will be explored in detail to show each step. It is important to note that even though lines $PS$ and $TV$ are actually parallel in the picture, a student, and therefore Kvasir, cannot assume that they are parallel unless this is explicitly stated in the problem description. This will be important to remember in the following sections.

![Diagram]

PS is parallel to TV.

Which of the following is true?
- TUR is supplementary to VUW.
- SQW is supplementary to VUR.
- PQR is supplementary to TUR.
- PQR is supplementary to VUW.
- SQR is supplementary to TUW.

Figure 3.1: Example Problem Solved by Kvasir
3.1 High-level Description

Before seeing in detail each part of Kvasir, it would be helpful to have a general idea of what happens. Figure 3.2 is a flowchart that shows the each part of the process and the flow of the data. To begin, the image is processed. Using computer vision techniques, each object in the figure is stored as raw image data. An analysis then proceeds on that data to create the knowledge base that contains logical declarations of lines, angles, and other objects in the image. Next, the problem description and question are read. Finally, each possible answer is tested against the knowledge base to determine which is the key and which are the distractors.

3.2 Inside the Black Box

While we refer to the black box as a system or an algorithm with inputs and outputs and simply expect that the process inside works properly, there are technologies used that should be covered. Sections 3.2.1 and 3.2.2 describe in detail the inner workings of the black box.

3.2.1 Figure Recognition

As was mentioned previously, one part of the black box is figure recognition. The figure recognition required for Kvasir concentrates upon recognizing the shapes and objects in the question. The object in the example shown in Figure 3.1 consists of two parallel lines and a transversal. Details of how the lines are recognized follows.
The initial work for this project was done by Gabe Mulley, an undergraduate at RPI. He used the Line Simplification Algorithm that was discovered in 1973 by David H. Douglas and Thomas K. Peucker with some slight modifications \cite{10}. The algorithm tries to represent an object as a series of points connected by straight lines. Initially, the endpoints of the object are used as the endpoints for the first line. Figure 3.3 displays this.

![Figure 3.3: Example application of the Line Simplification Algorithm developed by Douglas and Peucker. This is the initial step.](image)

This continues recursively until the distance to the line from the furthest point on the figure being approximated is less than a certain threshold. If it is greater than that threshold, a new point is created at the point that is the furthest away, creating two more lines for which the algorithm is again applied to.

![Figure 3.4: The Line Simplification Algorithm has split the figure into two parts.](image)

Figure 3.4 shows the image as a result of another recursion of the algorithm, with one half sufficiently approximated. In Figure 3.5 the image is shown as the algorithm recurses again, and in Figure 3.6 the algorithm is finished recursing since
the full figure has been approximated to the degree set by the threshold. Figure 3.7 shows the complete approximation, albeit a poor one because of the large threshold.

Figure 3.5: The figure as a result of another recursion of the algorithm.

Figure 3.6: Here the figure is split and approximated for the last time.

Figure 3.7: The final image is shown with both the approximations and the original figure.

This approach can be used to approximate figures that are straight lines or curved objects such as circles. In the example problem (Figure 3.1), it is very straightforward to approximate the lines, quite simply because they are lines. The endpoints of the real line are chosen for the approximation line but the distance between the approximation line and real line is zero. Mulley did make modifications to the algorithm to get it to work with closed loop figures such as those that intersect themselves.

MATLAB [20] was chosen as the system to develop in because of its extensive image processing toolkit. Its documentation is remarkably thorough while the
toolkit makes it very easy to read and write images in many formats, process individual pixels, and to then visualize what is being worked on. And, since the programming style is very similar to C or C++, it is immediately very easy to understand for one who has prior programming experience.

The aforementioned algorithm is applied to the images. All objects in the image are redefined as lines, closed loop objects such as circles and polygons are detected, and any intersections between objects are located. Even though the algorithm has simplified the objects to lines, it is unaware what this means. In the math example in Figure 3.1, PS, TV, and WR are all easily recognized as lines, yet stored as just X and Y locations. Also, the intersection point of Q is applied to both PS and WR and the intersection point of U is applied to TV and again to WR. This is then saved as a figure in MATLAB. But since each object is being redefined in terms of lines, objects such as circles do not appear the same as they once did. Many-sided polygons also often have a problem, but when dealing with just lines, the images look exactly the same as expected.

3.2.2 Optical Character Recognition

The other part of the black box is OCR. Kvasir uses the technology to label all the endpoints and intersections of the lines in Figure 3.1 or wherever else they might occur in other questions. Mulley found software that could be used for this purpose, but he never implemented it.

3.2.2.1 SimpleOCR

As we learned, there are many different OCR packages available for purchase to do this text extraction. However, many of these are expensive, well-developed tools that are more elaborate than what is needed for this simple case. Quite often they also do not have the option to interact via the command line, which was a requirement for Kvasir. After researching freely available alternatives, SimpleOCR was chosen because it did have a command line version that allowed Kvasir to call the executable directly. Upon reading the documentation, an additional command line option was discovered to give the program just a portion of the image. This was important, as previously the entire image was passed to the OCR.
software, which then returned all the text present. With all the text returned, it was unclear as to which character was actually labeling which point. So instead, with the command line option in use, when a character was discovered in the image, just the portion of the image that the OCR software needed to process was also passed to the program via its XY coordinates. This assured that each point was correctly labeled.

### 3.2.2.2 Tesseract

There were some difficulties with the SimpleOCR software, however. A minor problem was that it was only a demo version. This could have been easily solved by purchasing it, but the program was an executable compatible with only Windows machines. The RAIR Lab that produces Slate and Solomon attempts to be interoperable across platforms so the OCR software was required to be so as well.

Tesseract [19] is an open-source OCR engine currently at version 2.01 and has the source code freely available. It was initially developed at Hewlett-Packard from 1985 to 1995, was hosted at Source Forge for a period of time, until moving to Google Code where it is now found. The latest version of Tesseract comes with a Windows executable, but since the source code is provided, it can be compiled to work on Linux and OS X. Since it is open source and not a released product, it took some work to get it work properly in OS X.

Issues were discovered when using it on a PowerPC Mac versus an Intel-based Mac and with the character sets provided. Each of these bugs had a work-around, but a larger issue was the lack of the command line option that SimpleOCR had. This option, once discovered, was absolutely vital to guarantee that each point was correctly labeled. So instead a new image, the size of the character, is created and sent to the OCR engine. Even still, since the code for Tesseract is provided, there is a large amount of flexibility as to what can be done with it, including providing the command line option to process just a portion of the image.
3.3 Data Analysis

At this point, the image is no longer needed; all the visual information has been gathered from it. It is important to note that the processing done by the computer up until this point has not been making any interpretations on what is in the picture. While it may be simple for a computer to analyze a picture and see if two lines are parallel, that is something that a human would not be able to do, and even if one could, a student could not make that assumption unless the problem statement told them specifically to. To clearly illustrate this point, demonstrations have been accomplished using an image with two lines and one transversal with the two lines clearly not parallel to the human eye, yet since the problem description stated they were, the computer correctly answered the problem. Here though, the intelligent analysis a student would perform is described, regardless of how parallel the lines look.

First, the character labels are assigned to be the correct endpoints and intersections of the objects, since prior to this point, the labels are stored separately. The X and Y location of the label are compared to the endpoints and all the intersections. If they are within a certain margin, the object at that point is labeled.

The objects stored contain their physical locations, their labels, and the point of any intersections. Objects are then compared to determine which objects intersect each other, and from those intersections, several things can be determined. There are four angles that result from an intersection. Also, when there are intersections, there are four pairs of supplementary angles and two pairs of vertical, or opposite and congruent, angles.

The most difficult part of the analysis for the computer is examining the objects for what can be said about them. Again, this is a simple classification done by a human. To duplicate that, Kvasir does a form of pattern matching. The main object looked for is a line that intersects two other lines, or a transversal. The two other lines do not need to be parallel for it to be a transversal, but when they are, interesting properties are inherent of the set of lines. To easily label a transversal when the intersected lines are parallel, Kvasir calls the occurrence a hetch. An alternate pronunciation of the letter \( H \) in both Irish and Indian English,
it is designated thus because of the similarity between an H and the geometric formation. When a transversal is found (which does not require the lines to be parallel), the lines are analyzed to gather the correct starting points, endpoints, and intersecting points to generate the possibility of a hetch. The reason it is only a possibility is because only the problem description can provide the information that the lines are parallel. If there is no such information, then the assertion in SNARK which requires this will disregard the analysis that there may be a hetch.

However, when a hetch actually does occur, as covered previously in 2.5, each angle formed by the intersection of the lines has clear properties in relation to the other angles. So, as shown in Figure 3.8, the angles are labeled $a_1$ through $a_8$. Each $a_i$ will be later used by SNARK when trying to find the answer. Also, it is easier to use those labels when speaking about an angle in the hetch than to determine what the real name of the angle is.

![Figure 3.8: Template for a Hetch](image)

### 3.3.1 Output

Each line of output from MATLAB is a valid SNARK expression. As was detailed in section 2.2.2 before any form of line or angle can be read into SNARK, all constants must be declared. So this is the first step taken in MATLAB. This means that point $P$ and point $S$ need to be declared before line $PS$, for example.
For the example problem, the following shows all the declarations needed. Notice that names are not commutative, which means that both line PS and line SP need to be declared.

```scheme
(declare-constant 'R :sort 'point)
(declare-constant 'P :sort 'point)
(declare-constant 'Q :sort 'point)
(declare-constant 'S :sort 'point)
(declare-constant 'T :sort 'point)
(declare-constant 'V :sort 'point)
(declare-constant 'U :sort 'point)
(declare-constant 'W :sort 'point)
(declare-constant 'PS :sort 'line)
(declare-constant 'SP :sort 'line)
(declare-constant 'TV :sort 'line)
(declare-constant 'VT :sort 'line)
(declare-constant 'WR :sort 'line)
(declare-constant 'RW :sort 'line)
(declare-constant 'PQW :sort 'angle)
(declare-constant 'WQP :sort 'angle)
(declare-constant 'WQS :sort 'angle)
(declare-constant 'SQW :sort 'angle)
(declare-constant 'SQR :sort 'angle)
(declare-constant 'RQS :sort 'angle)
(declare-constant 'RQP :sort 'angle)
(declare-constant 'PQR :sort 'angle)
(declare-constant 'TUW :sort 'angle)
(declare-constant 'WUT :sort 'angle)
(declare-constant 'WUV :sort 'angle)
(declare-constant 'VUW :sort 'angle)
(declare-constant 'VUR :sort 'angle)
(declare-constant 'RUV :sort 'angle)
(declare-constant 'RUT :sort 'angle)
(declare-constant 'TUR :sort 'angle)
```

Once each label is declared to SNARK, objects using those labels can be asserted. Specifically, lines and angles are the objects being asserted, but relations need to be asserted as well. These relations include supplementary angles and parallel and intersecting lines. Below is the output for Figure 3.1.

```scheme
(assert '(= PS (line P S)))
(assert '(= SP (line S P)))
(assert '(= TV (line T V)))
(assert '(= VT (line V T)))
(assert '(= WR (line W R)))
```
As the analysis proceeds iteratively through all of the objects stored, the declarations and assertions are outputted as it goes. Yet throughout this, the analysis is saving objects to see if it can match them as hetches. In the example, at this point a hetch is found. Unlike lines and angles which have points to label them, hetches have none given by the image or problem description, so Kvasir assigns each a label. Each label is of the form $H_i$ where $i$ is an increasing number starting at 1; obviously in this case the hetch is declared as $H_1$. Once they are declared, the hetch is asserted with the lines that make it up. Each angle formed by the lines is then assigned the appropriate name, $a_1$ through $a_8$, as defined in Figure 3.8.
(assert '(= VUW (a5 h1)))
(assert '(= VUR (a6 h1)))
(assert '(= SQW (a7 h1)))
(assert '(= SQR (a8 h1)))

At this point, all of the raw image data has been represented in first-order logic as a set Φ of formulas. Each formula in Φ is a sentence and will be the knowledge base SNARK uses to determine which of the answers is the key.

### 3.4 Textual Information

One of the abilities of Slate is to translate text expressed in logically controlled English into multi-sorted logic (MSL). It can take this translated text and additional logic entered into Slate and then reason over this knowledge. This process can be automated or done manually by a human user. It can be then said that this process, while limited, gives Slate the ability to “read.” While the main purpose of this thesis has been to show the analysis of the visual aspect of the question, the textual information in the problem description and the actual question need to be brought into a logical form. Obviously, without knowing what question a user is asking, all the visual analysis is useless. So, the next part of this thesis will describe the means of converting the English into logic and the question processing.

### 3.4.1 Reading the Question

The method by which Slate reads revolves around the use of ACE. This method, shown in Figure 3.9, used with permission by Bringsjord et al.[4], can be described in three separate phases:

**Phase 1** Currently, Slate uses ACE’s form of controlled English, but the initial English texts must first be converted into ACE, following the rules it has set up. While CELT allows multiple meanings for a sentence, Slate requires a single meaning, which ACE provides, so that the English can be translated into formal logic. This conversion of English text can be automated[22] but is done at the moment by a human user.
Phase 2  Once in a proper ACE format, Slate uses the APE webservice to generate a discourse representation structure (DRS) from the controlled English. DRSs are basically a different way of writing or storing first-order-logic, and in this case are returned as XML, which makes it easy to process. They are important in discourse representation theory, which gives a method to represent the meaning of a text [17, 18].

Phase 3  Slate uses MSL entirely to represent and reason over the knowledge it is holding. Therefore, the DRSs are lastly translated into MSL.

Here, there are two things that need to be read: the problem description and the question. The problem description includes additional information that cannot be assumed from the figure, such as, in this case, that two lines are parallel. And the question, at this point, can only be a multiple choice question asking which of the answers is true — which makes the question become a test of the truth of each answer. So the problem description is read and inserted into the knowledge base for SNARK and the answers then are queries to SNARK if it can prove the answer. In the example problem, the problem description becomes:

(assert '(parallel PS TV))

The key and distractors (the possibly answers) become the following:

(prove '(supplementary TUR VUW))
(prove '(supplementary SQW VUR))
(prove '(supplementary PQR TUR))
(prove '(supplementary PQR VUW))
(prove '(supplementary SQR TUW))
3.5 Finding the Correct Answer

To find the correct answer, the output from reading the image and the problem description are used as the knowledge base against which each possible answer is tested. To determine which answer is correct, additional tacit knowledge of mathematical properties is required which the theorem prover uses to try to prove an answer correct. Several methods of storing this tacit knowledge are examined below.

3.5.1 Tarski’s Axioms

Alfred Tarski developed a set of 10 general axioms and one axiom schema that can be used to solve most mathematical problems involving Euclidean geometry\[32]\). To only need 10 axioms to find the answer to any geometric question is very appealing because the alternative would seem to be to laboriously encode every single rule of geometry that can be found in textbooks, such as rules about parallel lines and angles, to allow any geometric question. Of these alternatives, the first was more attractive.

Therefore attempt was made by a student in the RAIR lab to use just these 10 axioms and the schema that describes how they are to be used to answer the TIMSS questions. Below, the list of axioms can be seen (A1 through A10) along with the schema (A11). But first there are two relations that need explanation.

- \(ab \equiv cd\) means both that \(ab\) is congruent to \(cd\) and that the distance from \(a\) to \(b\) equals the distance from \(c\) to \(d\).

- \(a.b.c\) is a method for describing “betweeness;” that \(b\) is between \(a\) and \(c\) or that \(a, b, \) and \(c\) are collinear.

Assuming each variable is a point, the congruence relation gives this set of axioms the capability to allow one to measure the distance between points and to give them an orientation.

A1: \(ab \equiv ba\) Reflexivity

A2: \(ab \equiv pq \land ab \equiv rs \rightarrow pq \equiv rs\) Transitivity

A3: \(ab \equiv cc \rightarrow a = b\) Identity for equidistance
A4: \( \exists x \ (q.a.x \land ax \equiv bc) \) **Segment construction**

A5: \( (a \not= b \land a.b.c \land a'.b'.c' \land ab \equiv a'b' \land bc \equiv b'c' \land bd \equiv b'd' \land ad \equiv a'd') \rightarrow cd \equiv c'd' \) **Five-segment axiom**

A6: \( a.b.a \rightarrow a = b \) **Identity for betweenness**

A7: \( a.p.c \land b.q.c \rightarrow \exists x \ (p.x.b \land q.x.a) \) **Axiom of Pasch**

A8: \( \exists a \exists b \exists c \ (\neg a.b.c \land \neg b.c.a \land \neg c.a.b) \) **Lower dimension axiom**

A9: \( p \not= q \land ap \equiv aq \land bp \equiv bq \land cp \equiv cq \rightarrow a.b.c \lor b.c.a \lor c.a.b \) **Upper dimension axiom**

A10: \( a.d.t \land b.d.c \land a \not= d \rightarrow \exists x \exists y \ (a.b.x \land a.c.y \land x.t.y) \) **Euclid’s axiom**

A11: \( (\exists a \forall x \forall y \ (\phi(x) \land \psi(y) \rightarrow a.x.y)) \rightarrow (\exists b \forall x \forall y \ (\phi(x) \land \psi(y) \rightarrow x.b.y)) \) **Continuity axiom-schema**

An additional reason for using these axioms was that they are in first-order logic and little work was required to implement them in SNARK. However, this approach did not work. While the axioms are very general and therefore do not limit what can be answered, they take an extremely long time to prove the correct answer, and quite often never finding it.

### 3.5.2 Geometric Axioms

Hence, a second attempt to determining the correct answer was made using a different approach. This time, the problem was addressed with a much higher level technique. Instead of attempting to answer any geometric question, specific geometric theorems would be used that focused on the types of problems the students would encounter. This approach made more sense as a student would not be able to solve the problem using Tarski’s axioms and the computer was supposed to find an answer in the same way a human might. In fact, one textbook for teaching the course to those students was used to gather the theorems desired. In addition, this method made it easier to output the visual knowledge because it was something that was much more readable and understandable by a human.

To begin, several sorts, or types, were defined in SNARK:

```scheme
(declare-sort 'point)
(declare-sort 'line)
(declare-sort 'angle)
```
Points, lines, angles, and hetches are easily understood, and lastly a measure was defined that was used to hold the numeric measure of an angle. Each variable and label was declared to be one of these sorts, shown in Section 3.3.1 because the sorts were used in function and relation declarations. Given that the basic building blocks for defining objects are points, more complex objects were then declared as functions taking these points.

```
(declare-function 'line 2 :sort '(line point point))
(declare-function 'angle 3 :sort '(angle point point point))
```

As shown above, lines are declared as a function that takes two points and returns a line while angles are declared as a function that takes three points and returns an angle.

Each of these new functions has properties, such as commutativity, that are obvious and understood by a student but need to be explicitly programmed for SNARK to treat them the same way. This means that `(line A B)` is the same as `(line B A)`. What follows is how this looks in SNARK:

```
(assert '(forall (?point1 ?point2)
    (= (line ?point1 ?point2)
        (line ?point2 ?point1)))
  :name 'line-naming-is-commutative)
```

Yet having these types is not very useful unless there are relations using those types. So next, line intersections, transversals, and parallel lines were declared using the axioms from the textbook, with both parallel lines and intersections commutative since that is understood by a student.

```
(declare-relation 'intersects 2 :sort '(line line))
(declare-relation 'transversal 1 :sort '(line))
(declare-relation 'parallel 2 :sort '(line line))
```

Parallel lines and transversals have other requirements, however. In SNARK, transversals need there to be two lines to intersect:

```
(assert '(forall ((l1 :sort line))
    (iff (transversal l1)
        (exists ((l2 :sort line))
            (= (transversal l1)
                (transversal l2))))
  :name 'transversal-naming-is-commutative)
```

```
(declare-relation 'intersects 2 :sort '(line line))
(declare-relation 'transversal 1 :sort '(line))
(declare-relation 'parallel 2 :sort '(line line))
```

Parallel lines and transversals have other requirements, however. In SNARK, transversals need there to be two lines to intersect:
Parallel lines require that the two lines do not intersect each other:

\[
\text{(assert '(forall (?line1 ?line2)\\n\quad (iff (parallel ?line1 ?line2)\\n\quad \quad (and (not (intersects ?line1 ?line2))\\n\quad \quad \quad (not (= ?line1 line2))))))}\\n\text{:name 'definition-of-parallel)}
\]

Lastly, intersections create vertical and supplementary angles, both of which are commutative.

\[
\text{(declare-relation 'supplementary 2 :sort '(angle angle))}\\n\text{(declare-relation 'vertical 2 :sort '(angle angle))}
\]

In the case of vertical angles, they are congruent. To define two angles as congruent, it requires the use a measure function.

\[
\text{(declare-function 'measure 1 :sort '(measure angle))}\\n\text{(declare-relation 'congruent 2 :sort '(angle angle))}
\]

\[
\text{(assert '(forall (?angle1 ?angle2)\\n\quad (iff (congruent ?angle1 ?angle2)\\n\quad \quad (= (measure ?angle1)\\n\quad \quad \quad (measure ?angle2))))))}\\n\text{:name 'definition-of-congruency)}
\]

Pulling all of these sorts, relations, functions and assertions together allows one to be able to speak about hetches. Again, hetches are just two parallel lines and a transversal, which create other types of named angles: alternate interior angles (AIA), alternate exterior angles (AEA), and corresponding angles (CA). So, below, hetches are defined as a function taking three lines, and the second statement is a convenient way of stating that each of the AIA, AEA and CA angles all are relations about a pair of angles.

\[
\text{(declare-function 'hetch 3 :sort '(hetch line line line))}\\n\text{(dolist (fun '(AIA AEA CA))}\\n\quad \quad \text{(declare-relation fun 2 :sort '(angle angle)))}
\]
If two of the lines were parallel with one line intersecting them, the angles formed by
the intersection of the lines are created. Using the naming scheme in Figure 3.8, the
pairs $a_2$ and $a_6$ and $a_3$ and $a_7$ are alternate interior angles, the pairs $a_1$ and $a_5$ and
$a_4$ and $a_8$ are alternate exterior angles, and the pairs $a_1$ and $a_3$, $a_2$ and $a_4$, $a_5$ and
$a_7$, and $a_6$ and $a_8$ are corresponding angles. Each of these pairs of angles are also
congruent to each other and the relation is commutative. In order for SNARK to
be able to retrieve the actual angles from the hetch, each $a_i$ is defined as a function
that returns the real name of the angle.

\[
\text{(dolist (name '(a1 a2 a3 a4 a5 a6 a7 a8))}
  \text{(declare-function name 1 :sort '(angle hetch)))}
\]

At this point, the question can almost be answered by the machine but one
more assertion needs to be made. Four pairs of supplementary angles are created
by two lines intersecting, but when you have a hetch, since there are corresponding
angles, supplementary angles needed to be extended. This means that while supple-
mentary angles are typically adjacent to each other, the fact that the corresponding
angles are congruent makes more pairs of supplementary angles using angles on both
of the parallel lines. So, if $\angle 1$ and $\angle 2$ are supplementary, and $\angle 2$ and $\angle 3$ are congru-
ent, then $\angle 1$ and $\angle 3$ are also supplementary. What follows is how it is expressed in
SNARK:

\[
\text{(assert '(forall (?angle1 ?angle2)
  (implies (supplementary ?angle1 ?angle2)
    (forall (?angle3)
      (iff (congruent ?angle1 ?angle3)
        (supplementary ?angle2 ?angle3))))}
\]

Now, the image analysis has outputted all of the lines, angles, intersecting
lines, supplementary angles and the $a_1...a_8$ angles. If there was a pattern match for
a hetch, then that would have been outputted as well, although it would only work if
there was additional knowledge stating that the two intersected lines were parallel.
And, if the problem description and question was read, all that is left would be to
ask SNARK to prove each answer. If it is a distractor, SNARK will not be able
to find a proof because it is given a set time limit. But in the case of the correct
answer, or key, SNARK will quickly be able to find the right answer. The ensuing
statement shows the key in Figure 3.1 being tested in SNARK.
(prove '((supplementary SQW VUR))

3.5.3 Solved Problem

Here the full example question in Figure 3.1 follows. Each step and its output will be shown for the key and one distractor. First, the image is processed and analyzed in MATLAB by entering the following command:

rasterToVector ./images/transverse.bmp;

When it is finished, MATLAB will output the image data to image.txt:

```
(declare-constant 'R :sort 'point)
(declare-constant 'P :sort 'point)
(declare-constant 'Q :sort 'point)
(declare-constant 'S :sort 'point)
(declare-constant 'T :sort 'point)
(declare-constant 'V :sort 'point)
(declare-constant 'U :sort 'point)
(declare-constant 'W :sort 'point)
(declare-constant 'PS :sort 'line)
(declare-constant 'SP :sort 'line)
(declare-constant 'TV :sort 'line)
(declare-constant 'VT :sort 'line)
(declare-constant 'WR :sort 'line)
(declare-constant 'RW :sort 'line)
(declare-constant 'PQW :sort 'angle)
(declare-constant 'WQP :sort 'angle)
(declare-constant 'WQS :sort 'angle)
(declare-constant 'SQW :sort 'angle)
(declare-constant 'SQR :sort 'angle)
(declare-constant 'RQS :sort 'angle)
(declare-constant 'RQP :sort 'angle)
(declare-constant 'PQR :sort 'angle)
(declare-constant 'TUW :sort 'angle)
(declare-constant 'WUT :sort 'angle)
(declare-constant 'WUV :sort 'angle)
(declare-constant 'VUW :sort 'angle)
(declare-constant 'VUR :sort 'angle)
(declare-constant 'RUV :sort 'angle)
(declare-constant 'RUT :sort 'angle)
(declare-constant 'TUR :sort 'angle)
(assert '(= PS (line P S)))
(assert '(= SP (line S P)))
(assert '(= TV (line T V)))
(assert '(= VT (line V T)))
(assert '(= WR (line W R)))
(assert '(= RW (line R W)))
(assert '(intersects PS WR))
(assert '(= PQW (angle P Q W)))
(assert '(= WQP (angle W Q P)))
(assert '(= WQS (angle W Q S)))
(assert '(= SQW (angle S Q W)))
(assert '(= SQR (angle S Q R)))
(assert '(= RQS (angle R Q S)))
(assert '(= RQP (angle R Q P)))
(assert '(= PQR (angle P Q R)))
(assert '(= supplementary PQW QWS))
```
Also, the textual part of the question needs to be processed. It is processed by a call to Slate’s low-level process that takes ACE text and converts it first to DRS’s Section 3.4.1 and then to first-order logic. Both the problem description and the possible answers are converted separately.

At this point in time, SNARK will have the axioms it needs compiled from the file lines-and-angles.lisp. Before each answer can be tested, SNARK needs to be initialized, have the axioms setup, and the image and text data loaded. At the command line, the following function is executed:

(setup-load-data)

Of the possible answers, first a distractor is tested. Again at the command line, the following is entered:

(prove '(supplementary TUR VUW))

Since it is a distractor, it never finds a refutation so it runs until the time limit is exceeded. The attempt to find a refutation causes it to try many different formulas, so many that the full output cannot be shown. Instead, the summary that SNARK reports follows:

; Summary of computation:
; 9773 formulas have been input or derived (from 520 formulas).
; 4008 (41%) were retained. Of these,
43

; 126 (3%) were simplified or subsumed later,
; 387 (10%) were deleted later because the agenda was full,
; 3495 (87%) are still being kept.

; Run time in seconds excluding printing time:
; 1.052 7% Resolution (520 calls)
; 0.113 1% Paramodulation (520 calls)
; 0.287 2% Condensing (177 calls)
; 6.837 45% Forward subsumption (775 calls)
; 1.528 10% Backward subsumption (3666 calls)
; 0.523 3% Clause clause subsumption (11867 calls)
; 0.008 0% Ordering (420 calls)
; 0.007 0% Sortal reasoning (380 calls)
; 3.329 22% Other
; 15.054 Total

; Term-hash-array has 796 terms in all.
; Path-index has 567 entries (784 at peak, 845 added, 278 deleted).
; Tree-index has 540 nodes (660 at peak, 699 added, 159 deleted).
; Tree-index has 540 nodes (784 at peak, 845 added, 278 deleted).
; Retrieved 121605 generalization terms in 35663 calls.
; Retrieved 75496 instance terms in 9984 calls.
; Retrieved 12121 unifiable terms in 1835 calls.

; The agenda of other wffs to process has 51 entries:
; 48 with value (2 8) 3 with value (2 12)
; The agenda of everything else has 2999 entries:
; 2653 with value (4 8) 73 with value (4 11) 467 with value (4 12)
; 6 with value (4 9)

:RUN-TIME-LIMIT

At the end it prints :RUN-TIME-LIMIT which tells the user that it is a distractor.

Now the key is tested. Again, there are many formulas used by SNARK that will not be shown, but when the answer is found, a section is printed enclosed in
(Refutation ... ) where ... is the refutation. For the key
(prove '(supplementary SQW VUR))

the refutation and statistics from SNARK are shown below:

(Refutation
 (Row line-naming-is-commutative
  (= (line ?point ?point1) (line ?point1 ?point))
  assertion)
 (Row angle-naming-is-commutative
  (= (angle ?point ?point1 ?point2) (angle ?point2 ?point1 ?point))
  assertion)
 (Row supplementary-is-commutative
  (or (not (supplementary ?angle ?angle1)) (supplementary ?angle1 ?angle))
  assertion)
 (Row 15
  (or (not (supplementary ?angle ?angle1)) (not (congruent ?angle ?angle2))
  (supplementary ?angle1 ?angle2))
  assertion)
 (Row aia-aea-ca-are-commutative
  (or (not (ca ?angle ?angle1)) (ca ?angle1 ?angle))
  assertion)
 (Row aia-aea-ca-are-congruent
  (or (not (ca ?angle ?angle1)) (congruent ?angle ?angle1))
assertion)
(Row 31
(or (not (= ?hetch (hetch ?line ?line1 ?line2)))
(not (parallel ?line ?line2))
(not (intersects ?line ?line1))
(not (intersects ?line2 ?line1))
(ca (a7 ?hetch) (a5 ?hetch)))
assertion)
(Row 39
(= wr (line w r))
assertion)
(Row 40
(= rw (line r w))
assertion)
(Row 41
(intersects ps wr)
assertion)
(Row 54
(intersects tv wr)
assertion)
(Row 57
(= wuv (angle w u v))
assertion)
(Row 58
(= wuv (angle v u w))
assertion)
(Row 59
(= vur (angle v u r))
assertion)
(Row 60
(= ruv (angle r u v))
assertion)
(Row 64
(supplementary wuv vur)
assertion)
(Row 67
(= h1 (hetch ps rw tv))
assertion)
(Row 72
(= vuv (a5 h1))
assertion)
(Row 74
(= sqw (a7 h1))
assertion)
(Row 76
(parallel ps tv)
assertion)
(Row 77
(not (supplementary sqw vur))
negated_conjecture)
(Row 95
(not (supplementary vur sqw))
(resolve 77 supplementary-is-commutative))
(Row 99
(= ru wr)
(rewrite (paramodulate 40 line-naming-is-commutative) 39))
(Row 101
(intersects ps rw)
(rewrite 41 99))
(Row 102
(intersects tv rw)
(rewrite 54 99))
(Row 125
(= vuv vuv)
(rewrite (paramodulate 58 angle-naming-is-commutative) 57))
(Row 128
(supplementary vuw vur)
(rewrite 64 125))
(Row 131
  (= rvu vur)
  (rewrite (paramodulate 60 angle-naming-is-commutative) 59))
(Row 137
  (not (supplementary ruv sqw))
  (rewrite 95 131))
(Row 138
  (supplementary vuw ruv)
  (rewrite 128 131))
(Row 174
  (or (not (congruent vuw ?angle)) (supplementary ruv ?angle))
  (resolve 15 138))
(Row 359
  (ca sqw vuw)
  (rewrite (resolve 31 67) 72 74 102 101 76))
(Row 574
  (ca vuw sqw)
  (resolve aia-aea-ca-are-commutative 359))
(Row 604
  (congruent vuw sqw)
  (resolve aia-aea-ca-are-congruent 574))
(Row 697
  (supplementary ruv sqw)
  (resolve 174 604))
(Row 698
  false
  (rewrite 137 697))
)

; Summary of computation:
; 1034 formulas have been input or derived (from 188 formulas).
; 698 (68%) were retained. Of these,
; 45 ( 6%) were simplified or subsumed later,
; 653 (94%) are still being kept.
;
; Run time in seconds excluding printing time:
; 0.152 7% Resolution (187 calls)
; 0.025 1% Paramodulation (187 calls)
; 0.103 5% Condensing (586 calls)
; 0.620 27% Forward subsumption (586 calls)
; 0.353 16% Backward subsumption (528 calls)
; 0.089 4% Clause clause subsumption (1408 calls)
; 0.277 12% Forward simplification (986 calls)
; 0.036 2% Backward simplification (698 calls)
; 0.017 2% Ordering (420 calls)
; 0.006 0% Sortal reasoning (377 calls)
; 0.572 25% Other
; 2.270 Total
;
; Term-hash-array has 533 terms in all.
; Path-index has 521 entries (521 at peak, 581 added, 60 deleted).
; Path-index has 646 nodes (646 at peak, 683 added, 37 deleted).
; Tree-index has 521 entries (521 at peak, 581 added, 60 deleted).
; Tree-index has 796 nodes (796 at peak, 913 added, 117 deleted).
; Retrieved 15041 generalization terms in 9650 calls.
; Retrieved 17423 instance terms in 2935 calls.
; Retrieved 2543 unifiable terms in 686 calls.
;
; The agenda of other wffs to process has 15 entries:
; 2 with value (2 4) 1 with value (2 11) 4 with value (2 12)
; 8 with value (2 8)
; The agenda of everything else has 490 entries:
; 2 with value (4 4) 3 with value (4 18) 24 with value (4 31)
; 90 with value (4 8) 2 with value (4 19) 8 with value (4 64)
; 6 with value (4 9) 1 with value (4 20) 35 with value (4 68)
The final line, ':PROOF-FOUND', tells the user that it is the key to the question.
CHAPTER 4
Conclusions and Future Work

There is an immense amount of future work that could be done in this field. This thesis only covered one small area of the relevant mathematical world, and there is a myriad of more complex areas than this to be explored.

4.1 Next Steps

The obvious and immediate prospects would be to extend Kvasir to support additional types of mathematical problems. Figures 1.3 and 1.4 are TIMSS problems previously shown that involve calculations of angles using lines and angle properties and would be excellent areas in which to continue the research for Kvasir.

4.2 Further Into the Future

Closely related fields such as physics and statics would also have direct ties to this work. Mathematical in nature, both fields require a greater use of formulas while still needing the analysis that Kvasir already supports. The allowed kinds of questions could also be expanded beyond “Which of the following is true?” In these cases, answering a question would be much more difficult because of the work required in determining what the question is asking.

4.3 Long-term Vision

Work on Kvasir has been focused in the classroom environment in the area of geometry. The ultimate long-term vision is to have machines that are capable of “Learning by Seeing” in any domain, namely, the real world. At that point in time, the aforementioned questions posed to Google will be answerable, though it is unlikely they will still be asked in that fashion.
APPENDIX A
Glossary

Key
The correct answer in a multiple choice question.

Distractors
The incorrect answers in a multiple choice question.

Parallel Lines
Lines in the same plane that do not intersect. The symbol $\parallel$ means parallel.

Transversal
A line that intersects two lines to form eight angles.

Hetch
The term used in this thesis to describe the occurrence of a transversal of two parallel lines, since a transversal does not require the lines to be parallel. Figure A.1 shows the naming scheme ($a_1$ to $a_8$) that is used in this thesis.

![Hetch Naming Scheme](image)

Figure A.1: Hetch Naming Scheme

Congruent Angles
Two angles are congruent if they have the same measure. The symbol $\cong$ means is congruent to.

**Alternate Interior Angles**

In Figure A.2, transversal $t$ intersects lines $p$ and $q$. $\angle 2$ and $\angle 6$, $\angle 3$ and $\angle 7$ are alternate interior angles. If lines $p$ and $q$ are parallel, then these angles are congruent.

![Figure A.2: Parallel Lines and a Transversal](image)

**Alternate Exterior Angles**

In Figure A.2, transversal $t$ intersects $p$ and $q$. $\angle 1$ and $\angle 5$, $\angle 4$ and $\angle 8$ are alternate exterior angles. If lines $p$ and $q$ are parallel, then these angles are congruent.

**Corresponding Angles**

Angles that have the same position on two different parallel lines cut by a transversal. In Figure A.2, $\angle 1$ and $\angle 3$, $\angle 2$ and $\angle 4$, $\angle 5$ and $\angle 7$, and $\angle 6$ and $\angle 8$ are corresponding angles.

**Supplementary Angles**

Two angles are supplementary if the sum of their measures is $180^\circ$. In Figure A.3, $\angle 1$ and $\angle 2$, $\angle 2$ and $\angle 3$, $\angle 3$ and $\angle 4$, and $\angle 4$ and $\angle 1$ are supplementary angle pairs.
Figure A.3: Intersecting Lines

Vertical Angles

Congruent angles formed by the intersection of two lines. In Figure A.3, $\angle 1$ and $\angle 3$, $\angle 2$ and $\angle 4$ are vertical angles.
APPENDIX B

Code

Some of the following code for figure recognition was written by Gabe Mulley. Some modifications were needed for Kvasir to work properly, therefore one cannot separate the work done by Mulley and the work done for this thesis. For this reason, all of the code Kvasir uses has been included in this section.

B.1 MATLAB CODE

assignCharacterToPoint.m

function [characters, objects] = assignCharacterToPoint(objects, characters)

margin = 15;

for i = 1:length(characters)
    for j = 1:length(objects)
        x = objects{j}.start_p(2);
        y = objects{j}.start_p(1);
        if (x >= characters(i).x1 - margin && y >= characters(i).y1 ... - margin && x <= characters(i).x1 + characters(i).width ... + margin && y <= characters(i).y1 + characters(i).height + margin)
            characters(i).point_x = x;
            characters(i).point_y = y;
            objects{j}.start_char = characters(i).character{1};
        end

        x = objects{j}.end_p(2);
        y = objects{j}.end_p(1);
        if (x >= characters(i).x1 - margin && y >= characters(i).y1 ... - margin && x <= characters(i).x1 + characters(i).width ... + margin && y <= characters(i).y1 + characters(i).height + margin)
            characters(i).point_x = x;
            characters(i).point_y = y;
            objects{j}.end_char = characters(i).character{1};
        end

s = size(objects{j}.intersects);
for n = 1:s
    x = objects{j}.intersects(n,3);
    y = objects{j}.intersects(n,2);
    if (x >= characters(i).x1 - margin && y >= characters(i).y1 ... - margin && x <= characters(i).x1 + characters(i).width ...
+ margin && y <= characters(i).y1 + characters(i).height ...
+ margin)
            characters(i).point_x = x;
            characters(i).point_y = y;
            objects{j}.intersects(n,4) = characters(i).character{1};
        end
    end
end

boundingBox.m

function [box] = boundingBox(points)
    R = sort(points,1);
    height = abs(R(1,1)-R(end,1));
    width = abs(R(1,2)-R(end,2));
    box = [[R(1,1),R(1,2)];[R(end,1),R(end,2)];[height,width]];

circleIntersection.m

function roots = circleIntersection(x1,y1,r1,x2,y2,r2)
% C code found at:
% http://local.wasp.uwa.edu.au/~pbourke/geometry/2circle/tvoght.c
% public domain. Explanation at:
% http://local.wasp.uwa.edu.au/~pbourke/geometry/2circle/

% dx and dy are the vertical and horizontal distances between the
% circle centers
dx = x2 - x1;
 dy = y2 - y1;

% Determine the straight-line distance between the centers.
d = sqrt(dx^2 + dy^2);

if d > (r1 + r2)  % circles don’t intersect
    roots = [];
    return;
elseif d < abs(r1 - r2)  % one circle inside other
    roots = [];
    return;
end

% If a line is drawn between the centers of the two circles, a is the
% distance from the center of circle 1 to a point on the line, p,
% throughwhich a perpendiculr line could be drawn through the points
% of intersections

\[
a = \frac{(r1^2 - r2^2 + d^2)}{(2d)};
\]

\% The coordinates of p, which is distance a from x1,y1.
tempX = x1 + (dx \times a/d);
tempY = y1 + (dy \times a/d);

\[
h = \sqrt{r1^2 - a^2};
\]

\% The distance from p2 to either of the intersection points
rx = -dy \times (h/d);
ry = dx \times (h/d);

\% The coordinates of the intersection points.
roots.x(1) = tempX + rx;
roots.y(1) = tempY + ry;
if ry \neq 0
    roots.x(2) = tempX - rx;
    roots.y(2) = tempY - ry;
end

connected.m

function [P,num] = connected(OBJ,S,used)

P = [];
num = 0;

used = [S;used];

for c = -1:1
    for r = -1:1
        test_r = S(1,1)+r;
        test_c = S(1,2)+c;
        ok = isempty(find(used(:,1)==test_r & used(:,2)==test_c, 1));
        try
            if((OBJ(test_r, test_c)>0) && ok)
                P = [P; [test_r test_c]];
                num = num + 1;
            end
        catch
            end
        end
    end
end

createBoundingBox.m

function Box = createBoundingBox(array)
\% finds the region around the characters in the file.
% returns: Box that contains the x and y coordinates of the box and
% its height and width

s = size(array);
y_length = s(1);
x_length = s(2);
width = 17;
height = 15;
b = 1;
newBox = true;
moveBox = -1;

Box = [];

for i = 1:y_length
    for j = 1:x_length
        % check for first 0 you come to, then create box: make sure
        % first zero isn’t in box.
        if array(i,j) == 0
            for k = 1:length(Box)
                if (i <= Box(k).y1 + Box(k).height && j <= Box(k).x1 ...
                    + Box(k).width)
                    if (i >= Box(k).y1 && j >= Box(k).x1)
                        newBox = false;
                    elseif (Box(k).y1 - height < i && Box(k).x1 - width < j)
                        newBox = false;
                        moveBox = k;
                    end
                end
            end
            if newBox == true
                if(j > 2)
                    Box(b).x1 = j - 2;
                else
                    Box(b).x1 = 0;
                end
                if(i > 2)
                    Box(b).y1 = i - 2;
                else
                    Box(b).y1 = 0;
                end
                if(j + width > x_length)
                    Box(b).width = x_length - j + 2;
                else
                    Box(b).width = width;
                end
                if(i + height > y_length)
                    Box(b).height = y_length - i + 2;
                else
                    Box(b) = [];
                end
            end
        end
    end
end
Box(b).height = height;
end
b = b+1;
elseif moveBox ~= -1
% Only should be moving box left, not down.
if(j > 2)
    Box(moveBox).x1 = j - 2;
else
    Box(moveBox).x1 = 0;
end
if(j + width > x_length)
    Box(moveBox).width = x_length - j + 1;
else
    Box(moveBox).width = width;
end
end
end
newBox = true;
moveBox = -1;
end
detectIntersections.m

function [objects] = detectIntersections(objects,dx,dy)
if(nargin < 3), dy = 2; end
if(nargin < 2), dx = 2; end
for l1 = 1:(length(objects)-1)
    box1 = objects{l1}.boundingBox;
    box1(1,1) = box1(1,1)-dy; box1(2,1) = box1(2,1)+dy; box1(3,1) = ...
    box1(3,1)+(2*dy);
    box1(1,2) = box1(1,2)-dx; box1(2,2) = box1(2,2)+dx; box1(3,2) = ...
    box1(3,2)+(2*dx);
    if(isfield(objects{l1},'children'))
        obj1 = objects{l1}.children;
    else
        obj1 = objects{l1};
    end
    for o1 = 1:length(obj1)
        for l2 = (l1+1):length(objects)
            if(isfield(objects{l2},'children'))
                obj2 = objects{l2}.children;
            else
                obj2 = objects{l2};
            end
            for o2 = 1:length(obj2)
                box2 = obj2(o2).boundingBox;
            end
        end
    end
end
box2(1,1) = box2(1,1)-dy; box2(2,1) = box2(2,1)+dy; ...
box2(3,1) = box2(3,1)+(2*dy);
box2(1,2) = box2(1,2)-dx; box2(2,2) = box2(2,2)+dx; ...
box2(3,2) = box2(3,2)+(2*dx);
if(box1(1,1) <= box2(1,1))
tbox = box1;
bbox = box2;
else
tbox = box2;
bbox = box1;
end
if(bbox(1,:) <= tbox(2,:))
c = []; r = [];
warning off all;
a = pointOfIntersection(obj1(o1).eq, obj2(o2).eq);
warning on all;
if(isfield(a,'x'))
c = [c; round(double(a.x))];
r = [r; round(double(a.y))];
end
for n = 1:length(c)
if(all([r(n) c(n)] >= tbox(1,:)) && all([r(n) ... c(n)] <= tbox(2,:)) && all([r(n) c(n)] >= ... bbox(1,:) && all([r(n) ... c(n)] <= bbox(2,:))))
objects{l1}.intersects = ...
[objects{l1}.intersects; [12 r(n) c(n)]];
objects{l2}.intersects = ...
[objects{l2}.intersects; [11 r(n) c(n)]];
end
end
euDist.m

function dist = neuDist(p1,p2)
dist = sqrt((p2(1) - p1(1))^2 + (p2(2) - p1(2))^2);

exportLinesToFile.m

function [err] = exportLinesToFile(objects,characters)
err = [];

[characters, objects] = assignCharacterToPoint(objects, characters);
additional_output = ''; 
for i = 1:length(characters)
    fprintf(1, '(declare-constant \''c :sort \''point)\n', ... 
        characters(i).character{1});
end
% Output object types with names
for i = 1:length(objects)
    if(strcmp(objects{i}.type,'line'))
        if(isfield(objects{i}, 'start_char') && isfield(objects{i}, ... 
            'end_char'))
            fprintf(1, '(declare-constant \''c%c :sort \''line)\n', ... 
                objects{i}.start_char, objects{i}.end_char);
            fprintf(1, '(declare-constant \''c%c :sort \''line)\n', ... 
                objects{i}.end_char, objects{i}.start_char);
            a = sprintf('(assert \'(= %c%c (line %c %c)))\n', ... 
                objects{i}.start_char, objects{i}.end_char, ... 
                objects{i}.start_char, objects{i}.end_char);
            b = sprintf('(assert \'(= %c%c (line %c %c)))\n', ... 
                objects{i}.end_char, objects{i}.start_char, ... 
                objects{i}.end_char, objects{i}.start_char);
            additional_output = [additional_output, a, b];
        end
    end
end
str_angle_dec = '(declare-constant \''c%c%c :sort \''angle)\n';
str_angle_assert = '(assert \'(= %c%c%c (angle %c %c %c)))\n';
str_angle_supp = '(assert \'(supplementary %c%c%c %c%c%c))\n';
str_inter_assert = '(assert \'(intersects %c%c %c%c))\n';
% Find Intersections and Output Angles
for i = 1:length(objects)
    intersections1 = objects{i}.intersects;
    for j = 1:size(intersections1)
        for k = i+1:length(objects)
            intersections2 = objects{k}.intersects;
            for m = 1:size(intersections2)
                % compare the objects to see if they have the same
                % intersection point
                if(intersections1(j,3) == intersections2(m,3) && ... 
                    intersections1(j,2) == intersections2(m,2) && ...
                    intersections1(j,3) == intersections2(m,3) && ...
                    isfield(objects{i}, 'start_char') && ... 
                    isfield(objects{i}, 'end_char') && ... 
                    isfield(objects{k}, 'start_char') && ... 
                    isfield(objects{k}, 'end_char') && ... 
                    isempty(intersections1(j,4)))
                    line1c1 = objects{i}.start_char;
                    line1c2 = objects{i}.end_char;
                    line2c1 = objects{k}.start_char;
                    line2c2 = objects{k}.end_char;
                    ic = char(intersections1(j,4));
% Any type of object can intersect another.
line1c2 = sprintf(str_inter_assert, line1c1, ...
additional_output = [additional_output, s];
% Only two lines can create angles - look into
% polygon tho, and do it for it also.
if(strcmp(objects{i}.type,'line') && ...
strcmpt(strcmp(objects{k}.type,'line'))
% declare all angle names, such as ABC then also CBA
fprintf(1, str_angle_dec, line1c1, ic, line2c1);
fprintf(1, str_angle_dec, line2c1, ic, line1c1);
fprintf(1, str_angle_dec, line2c1, ic, line1c2);
fprintf(1, str_angle_dec, line1c2, ic, line2c1);
fprintf(1, str_angle_dec, line1c2, ic, line2c2);
fprintf(1, str_angle_dec, line2c2, ic, line1c2);
fprintf(1, str_angle_dec, line2c2, ic, line1c1);
fprintf(1, str_angle_dec, line1c1, ic, line2c2);
% assert all angle names are angles, eg. ABC followed by CBA
a1 = sprintf(str_angle_assert, line1c1, ...
ic, line2c1, line1c1, ic, line1c1);
a2 = sprintf(str_angle_assert, line2c1, ...
ic, line1c1, line2c1, ic, line1c1);
a3 = sprintf(str_angle_assert, line2c1, ...
ic, line1c2, line1c2, ic, line1c2);
a4 = sprintf(str_angle_assert, line1c2, ...
ic, line2c1, line1c2, ic, line2c1);
a5 = sprintf(str_angle_assert, line1c2, ...
ic, line2c2, line1c2, ic, line2c2);
a6 = sprintf(str_angle_assert, line2c2, ...
ic, line1c2, line2c2, ic, line1c2);
a7 = sprintf(str_angle_assert, line2c2, ...
ic, line1c1, line2c2, ic, line1c1);
a8 = sprintf(str_angle_assert, line1c1, ...
ic, line2c2, line1c1, ic, line2c2);
% Assert which angles are supplementary
s1 = sprintf(str_angle_supp, line1c1, ...
ic, line2c1, line2c1, ic, line1c2);
s2 = sprintf(str_angle_supp, line2c1, ...
ic, line1c2, line1c2, ic, line2c2);
s3 = sprintf(str_angle_supp, line1c2, ...
ic, line2c2, line2c2, ic, line1c1);
s4 = sprintf(str_angle_supp, line2c2, ...
ic, line1c1, line1c1, ic, line2c1);
additional_output = [additional_output, ...
a1, a2, a3, a4, a5, a6, a7, a8, s1, s2, s3, s4];
% Store in object what lines it intersects.
if(‘isfield(objects{i}, ’x_lines’)"
    objects{i}.x_lines(1) = {line2c1};
    objects{i}.x_lines(2) = {line2c2};
    objects{i}.x_points(1) = {ic};
else
    num_x_lines = length(objects{i}.x_lines);
    objects{i}.x_lines(num_x_lines + 1) = {line1c1};
    objects{i}.x_lines(num_x_lines + 2) = {line1c2};
    objects{i}.x_points(num_x_lines / 2 + 1) = {ic};
end
if(‘isfield(objects{k}, ’x_lines’)"
    objects{k}.x_lines(1) = {line1c1};
    objects{k}.x_lines(2) = {line1c2};
    objects{k}.x_points(1) = {ic};
else
    num_x_lines = length(objects{i}.x_lines);
    objects{k}.x_lines(num_x_lines + 1) = {line1c1};
    objects{k}.x_lines(num_x_lines + 2) = {line1c2};
    objects{k}.x_points(num_x_lines / 2 + 1) = {ic};
end
end
end
end
end
end
end
fprintf(‘%s’, additional_output);

hetch_count = 1;
for i = 1:length(objects)
    obj = objects{i};
    num_xs = length(obj.x_points);
    if(num_xs > 1)
        for j = 1:num_xs - 1
            % Sort the 4 points so that they go in ascending order if
            % the line isn’t vertical by x values
            if(obj.start_p(1) ~= obj.end_p(1))
                % Sort the start and end points, plus rearrange the
                % start and end characters
                if(obj.start_p(2) > obj.end_p(2))
                    temp = obj.start_p;
                    obj.start_p = obj.end_p;
                    obj.end_p = temp;
                    temp = obj.points(1,:);
                    obj.points(1,:) = obj.points(2,:);
                    obj.points(2,:) = temp;
                    temp = obj.start_char;
                    obj.start_char = obj.end_char;
                    obj.end_char = temp;
```
end

% Sort the 2 intersection points you are looking at
% and rearrange the intersection characters and the
% characters associated with the swapped lines
if(obj.intersects(j,3) > obj.intersects(j+1,3))
    temp = obj.intersects(j,:);
    obj.intersects(j,:) = obj.intersects(j+1,:);
    temp = obj.x_points{j};
    obj.x_points{j} = obj.x_points{j+1};
    temp = obj.x_lines{j*2-1};
    obj.x_lines{j*2-1} = obj.x_lines{(j+1)*2-1};
    temp = obj.x_lines{j*2};
    obj.x_lines{j*2} = obj.x_lines{(j+1)*2};
    end
else % if the line is vertical sort by y values instead
    if(obj.start_p(1) > obj.end_p(1))
        temp = obj.start_p;
        obj.start_p = obj.end_p;
        obj.end_p = temp;
        temp = obj.points(1,:);
        obj.points(1,:) = obj.points(2,:);
        obj.points(2,:) = temp;
        temp = obj.start_char;
        obj.start_char = obj.end_char;
        obj.end_char = temp;
    end
    % Sort the 2 intersection points you are looking at
    % and rearrange the intersection characters and the
    % characters associated with the swapped lines
    if(obj.intersects(j,2) > obj.intersects(j+1,2))
        temp = obj.intersects(j,:);
        obj.intersects(j,:) = obj.intersects(j+1,:);
        temp = obj.x_points{j};
        obj.x_points{j} = obj.x_points{j+1};
        temp = obj.x_lines{j*2-1};
        obj.x_lines{j*2-1} = obj.x_lines{(j+1)*2-1};
        temp = obj.x_lines{j*2};
        obj.x_lines{j*2} = obj.x_lines{(j+1)*2};
        end
end

if(obj.slope < 0)
    transc1 = obj.end_char;
    transc2 = obj.start_char;
ic1 = obj.x_points{j+1};
ic2 = obj.x_points{j};
line1c1 = obj.x_lines{(j+1)*2-1};
line1c2 = obj.x_lines{(j+1)*2};
line2c1 = obj.x_lines{j*2-1};
line2c2 = obj.x_lines{j*2};

else
    transc1 = obj.start_char;
    transc2 = obj.end_char;
ic1 = obj.x_points{j};
ic2 = obj.x_points{j+1};
line1c1 = obj.x_lines{j*2-1};
line1c2 = obj.x_lines{j*2};
line2c1 = obj.x_lines{(j+1)*2-1};
line2c2 = obj.x_lines{(j+1)*2};
end

fprintf(1, '(declare-constant \'H%i :sort \'hetch)\n', ...
hetch_count);
    fprintf(1, '(assert \'(= H%i (hetch %c%c %c%c %c%c)))\n', ...
hetch_count, line1c1, line1c2, transc1, transc2, ...
ic1, transc1, hetch_count);
        fprintf(1, '(assert \'(= %c%c%c (a1 h%i)))\n', line1c1, ...
ic1, transc1, hetch_count);
        fprintf(1, '(assert \'(= %c%c%c (a2 h%i)))\n', line1c1, ...
ic1, transc2, hetch_count);
        fprintf(1, '(assert \'(= %c%c%c (a3 h%i)))\n', line2c1, ...
ic2, transc1, hetch_count);
        fprintf(1, '(assert \'(= %c%c%c (a4 h%i)))\n', line2c1, ...
ic2, transc2, hetch_count);
        fprintf(1, '(assert \'(= %c%c%c (a5 h%i)))\n', line2c2, ...
ic2, transc2, hetch_count);
        fprintf(1, '(assert \'(= %c%c%c (a6 h%i)))\n', line2c2, ...
ic2, transc1, hetch_count);
        fprintf(1, '(assert \'(= %c%c%c (a7 h%i)))\n', line1c2, ...
ic1, transc2, hetch_count);
        fprintf(1, '(assert \'(= %c%c%c (a8 h%i)))\n', line1c2, ...
ic1, transc1, hetch_count);
    end
end

default:
    dx = end_p(1,2)-start_p(1,2);
dy = end_p(1,1)-start_p(1,1);

default

function [eq] = getModel(slope,point)
if(slope ~= Inf && slope ~= 0)
y_int = point(1) - slope*point(2);
if(y_int > 0)
    [eq,err] = sprintf('y=%1.3f*x+%1.3f',slope,y_int);
elseif(y_int == 0)
    [eq,err] = sprintf('y=%1.3f*x',slope);
elseif(y_int < 0)
    [eq,err] = sprintf('y=%1.3f*x%1.3f',slope,y_int);
end
elseif(slope == 0)
    [eq,err] = sprintf('y=%d',point(1));
else
    [eq,err] = sprintf('x=%d',point(2));
end

getNextPoint.m

function [adj,next,nslope] = getNextPoint(OBJ,cur,last,start_p,accuracy,sdiff)
nslope = 0;
next = [];
[adj,adj_r] = connected(OBJ,cur,last);
if(adj_r > 1)
    min_diff = Inf;
slope = 0;
    nslope = 0;
    next = [];
    for k = 1:adj_r
        [slope,diff] = getSlope(start_p, adj(k,:),cur);
        if(diff < min_diff)
            min_diff = diff;
            nslope = slope;
            if(diff < sdiff || accuracy < 20)
                % allow first two points to calculate average slope
                next = adj(k,:);
            end
        end
    end
elseif(adj_r == 1)
    [nslope,diff] = getSlope(start_p, adj(1,:),cur);
    next = adj(1,:);
else
    nslope = getSlope(start_p, cur);
end

getSlope.m
function [slope,diff] = getSlope(start_p,end_p,last_p)

compdiff = (nargin == 3);

dx = end_p(1,2)-start_p(1,2);
dy = end_p(1,1)-start_p(1,1);
if(dx ~= 0)
    slope = dy/dx;
else
    slope = Inf;
end

if(compdiff)
    dx = last_p(1,2)-start_p(1,2);
dy = last_p(1,1)-start_p(1,1);
if(dx ~= 0)
    old_slope = dy/dx;
else
    old_slope = Inf;
end
old_theta = atan(old_slope);
new_theta = atan(slope);
if(old_slope == Inf && slope == Inf)
    diff = 0;
elseif(old_slope == Inf)
    diff = min([abs(new_theta-old_theta),abs(new_theta+old_theta)]);
elseif(slope == Inf)
    diff = min([abs((-new_theta)-old_theta),abs(new_theta-old_theta)]);
else
    diff = abs(new_theta-old_theta);
end
end

groupObjects.m

function [objects] = groupObjects(lines)

[lines_r lines_c] = size(lines);
l1 = 1;
l2 = 1;
nobj = 1;
objects = {};
for l1 = 1:(lines_r-1)
    for l2 = (l1+1):lines_r
        [dist,lmat] = lineDist(lines(l1),lines(l2));
        if(dist <= 5)
            cp = lmat{1,2};
            if(~isempty(lines(l1).ends))
                [nend e_c] = find(((lines(l1).ends(:,1)-2)<=cp(1) & ... (lines(l1).ends(:,1)+2>=cp(1))) ...
& ((lines(l1).ends(:,2)-2)<=cp(2) ...
& (lines(l1).ends(:,2)+2)>=cp(2)))
if(isempty(nend))
    lines(l1).ends = [lines(l1).ends; cp];
end
else
    lines(l1).ends = [lines(l1).ends; cp];
end
if(~isempty(lines(l2).ends))
    [nend e_c] = find(((lines(l2).ends(:,1)-2)<=cp(1) & ...
& (lines(l2).ends(:,1)+2>=cp(1))) ...
& ((lines(l2).ends(:,2)-2)<=cp(2) ...
& (lines(l2).ends(:,2)+2)>=cp(2)));
    if(isempty(nend))
        lines(l2).ends = [lines(l2).ends; cp];
    end
else
    lines(l2).ends = [lines(l2).ends; cp];
end
lines(l1).connected = [lines(l1).connected; l2 cp];
lines(l2).connected = [lines(l2).connected; l1 cp];
end
end
% handle isolated lines and line chains
l = 1;
while l <= lines_r
    [ncp e_c] = size(lines(l).ends);
    if(ncp < 2)
        lines(l).boundingBox = boundingBox(lines(l).points);
        conn = lines(l).connected;
        if(~isempty(conn))
            [cr cc] = size(conn);
            for c = 1:cr
                l1 = conn(c,1);
                [cl cc] = find(lines(l1).connected(:,1)==l);
                cp = [lines(l1).connected(cl,2),lines(l1).connected(cl,3)];
                lines(l1).connected(cl,:) = [];
                [el ec] = find(lines(l1).connected(:,2)==cp(1) & ...
                lines(l1).connected(:,3)==cp(2));
                if(isempty(el))
                    % no more connections on this endpoint
                    [el ec] = find(lines(l1).ends(:,1)==cp(1) & ...            
                lines(l1).ends(:,2)==cp(2));
                    lines(l1).ends(el,:) = [];
                end
            end
        end
    objects{nobj,1} = lines(l);
nobj = nobj + 1;
lines(l).ends = [[0 0];[0 0]];
l = 0;
end
l = l + 1;
end

% closed loop handling
l_r = 0;
for l = 1:lines_r
if(~isequal(lines(l).ends,[[0 0];[0 0]]))
l_r = l_r + 1;
end
end
while l_r > 0
cur = 1;
while (isequal(lines(cur).ends,[[0 0];[0 0]]))
cur = cur + 1;
end
start_l = lines(cur).ends;
children = [];
points = [];
while(cur ~= 0)
lines(cur).boundingBox = boundingBox(lines(cur).points);
children = [children; lines(cur)];
points = [points;lines(cur).start_p];
conn = lines(cur).connected;
[ch_l cc_l] = size(conn);
ends = lines(cur).ends;
hasnt_looped = isempty(find((ends(:,1)==start_l(1,1) & ...
ends(:,2)==start_l(2,1)) | ...
ends(:,1)==start_l(1,2)) & ...
ends(:,2)==start_l(2,2))));
[chil_r chil_c] = size(children);
if(~isempty(conn) && (hasnt_looped || chil_r < 3))
min_diff = Inf;
[ch_l cc_l] = size(conn);
if(cl > 1)
for c = 1:cl
l1 = conn(c);
[dist,lmat] = lineDist(lines(cur),lines(l1));
[sl, diff] = getSlope(lmat{1,1},lmat{2,2},lmat{1,2});
if(diff < min_diff)
next = l1;
min_diff = diff;
end
end
else
l1 = conn(1);
next = l1;
end
else
next = 0;
end
lines(cur).ends = [[0 0];[0 0]];
[cr cc] = size(lines(cur).connected);
for c = 1:cr
  l1 = conn(c);
  [cl cc] = find(lines(cur).connected(:,1)==l1);
  lines(cur).connected(cl,:) = [];
  [cl cc] = find(lines(l1).connected(:,1)==cur);
  lines(l1).connected(cl,:) = [];
end
cur = next;
l_r = l_r - 1;
end
bb = boundingBox(points);
center_p = [round(bb(1,1)+(bb(3,1)/2)),round(bb(1,2)+(bb(3,2)/2))];
nsides = length(children);
if(nsides > 9)
  r = bb(3,1)/2;
  a = pi*(r^2);
  [eq,err] = sprintf('(x-%i)^2 + (y-%i)^2 = %1.3f', center_p(2), ...
  center_p(1), r^2);
  center_p(1), r^2);
  if(~isempty(err))
    disp('an error has occured');
  end
  circle = struct('type','circle','center',center_p, ...
  'radius',r,'area',a,'eq',eq,'boundingBox', bb, ...
  'points',points,'intersects',[]);
  objects{nobj,1} = circle;
  nobj = nobj + 1;
else
  a = polyarea(points(:,2),points(:,1));
  polygon = struct('type','polygon','center',center_p, ...
  'sides',nsides,'area',a,'boundingBox',bb, ...
  'children',children,'points',points,'intersects',[]);
  objects{nobj,1} = polygon;
  nobj = nobj + 1;
end
end

lineDist.m

function [min_dist,lmat] = lineDist(line1,line2)

lines = [[line1.start_p;line1.end_p],[line2.start_p;line2.end_p]];
min_dist = Inf;
for t1 = 1:2
  for t2 = 1:2
    dist = eucDist([lines(t1,1),lines(t1,2)],[lines(t2,3),lines(t2,4)]);
    if(dist < min_dist)
min_dist = dist;
start_p1 = [lines(mod(t1,2)+1,1),lines(mod(t1,2)+1,2)];
end_p1 = [lines(t1,1),lines(t1,2)];
start_p2 = [lines(t2,3),lines(t2,4)];
end_p2 = [lines(mod(t2,2)+1,3),lines(mod(t2,2)+1,4)];

if(nargout > 1)
    lmat = {start_p1, end_p1;start_p2, end_p2};
end

mergeBroken.m

function [merged] = mergeBroken(lines)

[lines_r lines_c] = size(lines);

l1 = 1;
while(l1 <= (length(lines)-1))
    l2 = l1 + 1;
    while(l2 <= length(lines))
        [dist,lmat] = lineDist(lines(l1,1),lines(l2,1));
        [nslope,sdiff] = getSlope(lmat{1,1},lmat{2,2},lmat{1,2});
        if(sdiff <= .1 && dist <= 5)
            eq = getModel(nslope,lmat{1,1});
            new = struct('type','line','points',lmat{1,1};...
                          lmat{2,2}),'start_p',lmat{1,1},'end_p',lmat{2,...
                          2},'slope',nslope,'eq',eq,'intersects',[],...
                          'boundingBox',[],'connected',[],'ends',[]);
            lines(l1) = new;
            lines(l2) = [];
        end
    l2 = l2 + 1;
end
    l1 = l1 + 1;
end
merged = lines;

objcSelect.m

function [I,points] = objselect(I,r,c,sdiff)
% Arguments
% I: image from which to extract the object
% r: row which contains the starting pixel
% c: column which contains the starting pixel
% pix: # of pixels to look ahead on a path to determine it's
% correctness
% sdiff: maximum difference in slope from the average before a point
% is considered incorrect

% Returns
% I: input image after object extraction
% points: array of points that completely describe the object

% Summary of Purpose
% This function removes objects from a picture and returns them as a
% coordinate vector. Each pixel is removed as it is processed,
% enabling the first curve to be recognized in it's entirety and any
% intersecting curves to be broken. Broken curves are later
% restored.

start_p = [r c]; % starting point
points = []; % an array of [r c] coordinates that
% describes the object

adj = [r c]; % adjacent pixels, see getNextPoint
cur = [r c]; % the pixel currently being processed
last = [-1 -1]; % the last pixel to be analyzed
[adj_r adj_c] = size(adj); % intialize adj_r, the number of adjacent
% pixels

while(adj_r >= 1 || isequal(cur,start_p) || isequal(last,start_p))
    [point_r point_c] = size(points);
    if(point_r > 10)
        start_p = points(point_r - 10,:);
    end
    [adj,next,slope] = getNextPoint(I,cur,last,start_p,point_r,sdiff);

    [adj_r adj_c] = size(adj);
    if(adj_r <= 2)
        I(cur(1),cur(2)) = 0;
    else
        I(cur(1),cur(2)) = adj_r;
    end

    points = [points; cur];
    last = cur;
    if(~isempty(next)),cur = next; else, break; end
end

pointOfIntersection.m

function point = pointOfIntersection(eq1, eq2)
vertLine1 = sscanf(eq1, 'x=%f');
vertLine2 = sscanf(eq2, 'x=%f');

line1 = sscanf(eq1, 'y=%f*x%f');
if length(line1) == 1  % Handles case where no y-intercept
    m1 = line1(1);
    b1 = 0;
elseif length(line1) == 2  % line in form y=mx+b
    m1 = line1(1);
    b1 = line1(2);
end

line2 = sscanf(eq2, 'y=%f*x%f');
if length(line2) == 1  % Handles case where no y-intercept
    m2 = line2(1);
    b2 = 0;
elseif length(line2) == 2  % line in form y=mx+b
    m2 = line2(1);
    b2 = line2(2);
end

horizLine1 = sscanf(eq1, 'y=%f');  % Horizontal Line
if ~isempty(horizLine1) && isempty(findstr(eq1,'*x'))
    m1 = 0;
    b1 = horizLine1(1);
    line1 = 1;
end

horizLine2 = sscanf(eq2, 'y=%f');  % Horizontal Line
if ~isempty(horizLine2) && isempty(findstr(eq2,'*x'))
    m2 = 0;
    b2 = horizLine2(1);
    line2 = 1;
end

circle1 = sscanf(eq1, 'x^2 + y^2 = %f');
if isempty(circle1)
circle1 = sscanf(eq1, 'x^2 + (y%f)^2 = %f');
if length(circle1) ~= 2
    circle1 = sscanf(eq1, '(x%f)^2 + y^2 = %f');
    if length(circle1) ~= 2
        circle1 = sscanf(eq1, '(x%f)^2 + (y%f)^2 = %f');
        if length(circle1) ~= 3
            circle1VarCount = -1;
        else
            circle1VarCount = 3;
            x1 = -circle1(1);
            y1 = -circle1(2);
            r1Squared = circle1(3);
        end
    else
        circle1VarCount = 2;
    end
else
    circle1VarCount = 2;
end
x1 = -circle1(1);
y1 = 0;
r1Squared = circle1(2);
end
else
circle1VarCount = 2;
x1 = 0;
y1 = -circle1(1);
r1Squared = circle1(2);
end
else
circle1VarCount = 1;
x1 = 0;
y1 = 0;
r1Squared = circle1(1);
end

circle2 = sscanf(eq2, 'x^2 + y^2 = %f');
if isempty(circle2)
circle2 = sscanf(eq2, 'x^2 + (y%f)^2 = %f');
if length(circle2) ~= 2
circle2 = sscanf(eq2, '(x%f)^2 + y^2 = %f');
if length(circle2) ~= 2
circle2 = sscanf(eq2, '(x%f)^2 + (y%f)^2 = %f');
if length(circle2) ~= 3
circle2VarCount = 3;
else
circle2VarCount = 2;
x2 = -circle2(1);
y2 = -circle2(2);
r2Squared = circle2(3);
end
else
circle2VarCount = 2;
x2 = -circle2(1);
y2 = 0;
r2Squared = circle2(2);
end
else
circle2VarCount = 2;
x2 = 0;
y2 = -circle2(1);
r2Squared = circle2(2);
end
else
circle2VarCount = 1;
x2 = 0;
y2 = 0;
r2Squared = circle2(1);
end

if length(vertLine1) == 1 && length(vertLine2) == 1
% Identical Vertical Lines
if vertLine1 == vertLine2
    point = vertLine1;
else
    point = struct([]);
end
elseif length(vertLine1) == 1 && length(line2) ~= 0
    % Vertical Line and Line intersection
    point.x = vertLine1;
    point.y = m2*vertLine1 + b2;
elseif length(line1) ~= 0 && length(vertLine2) == 1
    % Line and Vertical Line intersection
    point.x = vertLine2;
    point.y = m1*vertLine2 + b1;
elseif length(vertLine1) == 1 && circle2VarCount ~= -1
    % Vertical Line and Circle
    if abs(vertLine1) > x2 + sqrt(r2Squared)
        point = struct([]);
        return;
    end
    a = 1;
    b = 2*y2;
    c = (vertLine1-x2)^2 - r2Squared;
    yRoot = solveQuadratic(a,b,c);
    if isempty(yRoot)
        point = struct([]);
    elseif length(yRoot) == 1
        point.x = vertLine1;
        point.y = yRoot;
    else
        point.x(1) = vertLine1;
        point.x(2) = vertLine1;
        point.y(1) = yRoot(1);
        point.y(2) = yRoot(2);
    end
elseif circle1VarCount ~= -1 && length(vertLine2) == 1
    % Circle and Vertical Line
    if abs(vertLine2) > x1 + sqrt(r1Squared)
        point = struct([]);
        return;
    end
    a = 1;
    b = 2*y1;
    c = (vertLine2-x1)^2 - r1Squared;
    yRoot = solveQuadratic(a,b,c);
    if isempty(yRoot)
        point = struct([]);
    elseif length(yRoot) == 1
        point.x = vertLine2;
        point.y = yRoot;
else
    point.x(1) = vertLine2;
    point.x(2) = vertLine2;
    point.y(1) = yRoot(1);
    point.y(2) = yRoot(2);
end
elseif length(line1) ~= 0 && length(line2) ~= 0
    % 2 lines intersecting
    m = m1 - m2;
    bSum = b2 - b1;

    % Parallel Lines don't intersect
    if m == 0
        point = struct([]);
        return;
    end

    point.x = bSum / m;
    point.y = m1*point.x + b1;
elseif length(line1) ~= 0 && circle2VarCount ~= -1 % 1 line, 1 circle
    % Take the equation for the line which is in form y=mx+b, and put
    % it in the equation for the circle. Solve for x. Find y putting
    % x back into the line equation and solving for y.

    a = 1 + m1^2;
    b = 2*x2 + 2*(b1-y2);
    c = x2^2 + (b1-y2)^2 - r2Squared;
    xRoot = solveQuadratic(a,b,c);
    if isempty(xRoot) % No Intersections
        point = struct([]);
    elseif length(xRoot) == 1 % 1 Intersection
        point.x = xRoot;
        point.y = m1*xRoot + b1;
    else % 2 Intersections
        point.x = xRoot;
        point.y(1) = m1*xRoot(1) + b1;
        point.y(2) = m1*xRoot(2) + b1;
    end
elseif length(line2) ~= 0 && circle1VarCount ~= -1 % 1 circle, 1 line
    % Take the equation for the line which is in form y=mx+b, and put
    % it in the equation for the circle. Solve for x. Find y putting
    % x back into the line equation and solving for y.

    a = 1 + m2^2;
    b = 2*x1 + 2*(b2-y1);
    c = x1^2 + (b2-y1)^2 - r1Squared;
    xRoot = solveQuadratic(a,b,c);
    if isempty(xRoot) % No Intersections
        point = struct([]);
    elseif length(xRoot) == 1 % 1 Intersection
        point.x = xRoot;
        point.y = m2*xRoot + b2;
else % 2 Intersections
    point.x = xRoot;
    point.y(1) = m2*xRoot(1) + b2;
    point.y(2) = m2*xRoot(2) + b2;
end
elseif circle1VarCount ~= -1 && circle2VarCount ~= -1 % 2 circles
    r1 = sqrt(r1Squared);
    r2 = sqrt(r2Squared);
    point = circleIntersection(x1,y1,r1,x2,y2,r2);
end

processText.m

function characters = processText(img)
    % scans the picture to find the locations of all the characters and
    % then processes them with the ocr software to identify them

text = ~(img == 0);

    % check to see if there are no elements that are 0 in array - which is
    % black text
    if ~(any(text(:) == 0))
        characters = [];
        return;
    end

    characters = createBoundingBox(text);

    if ispc
        executableFileName = '.\tesseract\tesseract.exe';
        imageFileName = '.\textFromImage.bmp';
        textFileName = '.\textFromImage.txt';
        executableLine = [executableFileName,' ',imageFileName,' ',textFileName];
    else
        executableFileName = '/usr/local/bin/tesseract';
        imageFileName = './textFromImage.bmp';
        textFileName = './textFromImage.txt';
        executableLine = [executableFileName,' ',imageFileName,' ',textFileName];
    end

    for i = 1:length(characters)
        singleCharacter = imcrop(text,[characters(i).x1, characters(i).y1, ...
            characters(i).width, characters(i).height]);
        imwrite(singleCharacter, imageFileName, 'bmp');

        [result message] = system(executableLine);
        if(result == 0) % return value of 0 is good
            fid = fopen(textFileName,'r');
            if (fid == -1)
characters(i).character = []; fprintf(2, '%s', message);
else
    character = textscan(fid,'%c'); fclose(fid);

    % checks to see if character returned is empty or ''
    if(~isempty(character{1}))
        characters(i).character = character;
    else
        characters(i).character = [];
    end
end

else
    characters(i).character = [];
    if ispc
        fid = fopen('.tesseract.log','r');
        while 1
            tline = fgetl(fid);
            if ~ischar(tline)
                break
            end
            fprintf(2,'%s\n',tline);
        end
        fprintf(2, '\n'); fclose(fid);
    end
end
delete(imageFileName);
delete(textFileName);

rasterToVector.m

function [V] = rasterToVector(path,ldist,sdiff)
% Arguments
% path: string path to image to convert
% ldist: threshold for DP algorithm
% sdiff: maximum difference in slope from the average before a point
% is considered incorrect

% Returns
% Analyzed image of what was "read" in logic form.

% Summary of Purpose
% For each object in the picture, seperate it and vectorize it, then
% do some basic analysis on the results, finding intersection points
% and relationships between objects.
% Bugs
%   Edge of image bug
%   CLI look into

if nargin < 3, sdiff = 0.02; end
if nargin < 2, ldist = 2; end
if nargin < 1
    error('Pass as argument a file to read from.');
end

n = 1;
objects = {[]};
nodes = {[]};

I_full = imread(path);
characters = processText(I_full);
I = (I_full == 73);
[rows cols] = size(I);
T = bwmorph(I,'thin',Inf);
T = uint8(T);
for c = 1:cols
    for r = 1:rows
        % start at the upper left most pixel in the image
        if(T(r,c) == 1)
            [adj,adj_r] = connected(T,[r c],[1]);
            if(adj_r > 1 & T(r+1,c))
                continue;
            end
            % extract the object (line or arc)
            [T,obj] = objselect(T,r,c,sdiff);
            [obj_r obj_c] = size(obj);
            if(obj_r > 1)
                warning off all
                [node_loc] = split(obj,1,length(obj),ldist);
                warning on all
                nodes{1,n} = node_loc;
                objects{1,n} = obj;
                n = n+1;
            end
        end
    end
end

[obj_r obj_c] = size(objects);
lines = [];
for j = 1:obj_c
    n = nodes{1,j};
[n_r n_c] = size(n);
for k = 1:(n_r-1)
slope = getSlope(n(k,:),n(k+1,:));
eq = getModel(slope,n(k,:));
len = sqrt((n(k,1)-n(k+1,1))^2 + (n(k,2)-n(k+1,2))^2);
endpoints = sortrows([n(k,:);n(k+1,:);,[2 1]]);
if(len > 2)
l = struct('type','line','points',endpoints,'start_p', ...
endpoints(1,:),'end_p',endpoints(2,:), ...
'slope',slope,'eq',eq,'intersects',[], ... 
'boundingBox',[],'connected',[],'ends',[]);
lines = [lines;l];
end
end

% merge lines that have similar slope and are separated by a small gap
lines = mergeBroken(lines);

% detects closed loop objects (circles, polygons)
objects = groupObjects(lines);
% locate intersections between objects
objects = detectIntersections(objects);
lcolors = jet(length(objects));
fig2 = figure;
iptsetpref('ImshowBorder','tight');
imshow(false(size(T)));
for l = 1:length(objects)
  if(strcmp(objects{l}.type,'line'))
    line([objects{l}.start_p(2);objects{l}.end_p(2)], ... 
    [objects{l}.start_p(1);objects{l}.end_p(1)],'Color',lcolors(l,:))
    text(objects{l}.start_p(2)-5,objects{l}.start_p(1)-5, ... 
    num2str(l), 'FontSize', 8, 'Color', lcolors(l,:));
  elseif(strcmp(objects{l}.type,'circle'))
    text(objects{l}.boundingBox(1,2)-2, ... 
    objects{l}.boundingBox(1,1)-2,num2str(l), 'FontSize', ... 
    8, 'Color', lcolors(l,:));
    rectangle('Position', [objects{l}.boundingBox(1,2), ... 
    objects{l}.boundingBox(1,1), ... 
    objects{l}.boundingBox(3,2), ... 
    objects{l}.boundingBox(3,1)],'EdgeColor', ... 
    lcolors(l,:),'Curvature',[1,1]);
  elseif(strcmp(objects{l}.type,'polygon'))
    nlines = length(objects{l}.children);
    for n = 1:nlines
      line([objects{l}.children(n).start_p(2); ... 
      objects{l}.children(n).end_p(2)], ... 
      [objects{l}.children(n).start_p(1); ...
Export the objects and characters to a file.

```matlab
exportLinesToFile(objects, characters);
```

### solveQuadratic.m

The function `solveQuadratic` calculates the roots of a quadratic equation.

```matlab
function roots = solveQuadratic(a, b, c)
    discriminant = b^2 - 4 * a * c;
    if discriminant < 0
        roots = [];  % imaginary roots, no intersections
    elseif discriminant == 0
        roots = -b / (2 * a);  % 1 real root, one tangent intersection
    else
        roots(1) = (-b - sqrt(discriminant)) / (2 * a);
        roots(2) = (-b + sqrt(discriminant)) / (2 * a);  % 2 real roots, two intersections
    end
end
```

### split.m

The function `split` calculates the midpoint between two points.

```matlab
function [nodes] = split(points, start_i, end_i, thresh)
    start_p = points(start_i,:);
    end_p = points(end_i,:);
    max_dist = 0;
    max = 0;
    [r_p c_p] = size(points);
    for k = (start_i+1):end_i
        point = points(k,:);
        [dx_model dy_model] = getDiffs(start_p, end_p);
        [dx_point dy_point] = getDiffs(start_p, point);
        a_model = atan(dy_model/dx_model);
        a_point = atan(dx_point/dy_point);
        theta = (pi/2) - a_model - a_point;
        h_dist = eucDist(start_p, point);
```
\[ \text{dist} = \text{abs(hdist*\text{sin(theta)})}; \]

\[
\text{if} (\text{dist} \geq \text{max_dist}) \\
\quad \text{max_dist} = \text{dist}; \\
\quad \text{max} = k; \\
\text{end}
\]

\[
\text{end}
\]

\[
\text{if} (\text{max_dist} < \text{thresh}) \\
\quad \text{nodes} = \text{end}_p; \\
\text{else} \\
\quad [n1] = \text{split(points, start_i, max, thresh}); \\
\quad [n2] = \text{split(points, max, end_i, thresh}); \\
\quad \text{nodes} = [n1; n2]; \\
\text{end}
\]

\[
\text{end}
\]

\[
\text{if}(\text{start_i} == 1 \&\& \text{end_i} == \text{length(points)}) \\
\quad \text{nodes} = [\text{start}_p; \text{nodes}];
\]

\[
\text{end}
\]

## B.2 SNARK Code

*lines-and-angles.lisp*

```lisp
(in-package :snark-user)

(defun setup-snark ()
  (initialize)
  (run-time-limit 15)
  (use-resolution t)
  (use-paramodulation t))

(defun setup-sorts ()
  (declare-sort 'point)
  (declare-sort 'line)
  (declare-sort 'angle)
  (declare-sort 'measure)
  (declare-sort 'hetch)

  (declare-function 'line 2 :sort '(line point point))
  (declare-function 'angle 3 :sort '(angle point point point))
  (declare-relation 'intersects 2 :sort '(line line))
  (declare-relation 'transversal 1 :sort '(line))
  (declare-function 'measure 1 :sort '(measure angle))
  (declare-relation 'congruent 2 :sort '(angle angle))
  (declare-relation 'parallel 2 :sort '(line line))
  (declare-function 'hetch 3 :sort '(hetch line line line))
  (dolist (name '(a1 a2 a3 a4 a5 a6 a7 a8))
    (declare-function name 1 :sort '(angle hetch)))
```
(dolist (fun '(AIA AEA CA))
  (declare-relation fun 2 :sort '(angle angle)))
(declare-relation 'supplementary 2 :sort '(angle angle))
(declare-relation 'vertical 2 :sort '(angle angle))
)

(defun setup-geometry ()
  ;; Name (of line)
  (assert '(forall (?point1 ?point2)
                 (= (line ?point1 ?point2)
                     (line ?point2 ?point1)))
   :name 'line-naming-is-commutative)

  ;; Name (of angle)
  (assert '(forall (?point1 ?point2 ?point3)
                 (= (angle ?point1 ?point2 ?point3)
                     (angle ?point3 ?point2 ?point1)))
   :name 'angle-naming-is-commutative)

  ;; Intersection ................modify this to not work with just lines.
  (assert '(forall ((l1 :sort line) (l2 :sort line))
               (iff (intersects l1 l2)
                    (intersects l2 l1)))
   :name 'intersection-is-commutative)

  ;; Transversal
  (assert '(forall ((l1 :sort line))
               (iff (transversal l1)
                    (exists ((l2 :sort line) (l3 :sort line))
                        (and (not (= l2 l3))
                             (intersects l1 l2)
                             (intersects l1 l3)))
   :name 'definition-of-transversal)

  ;; Congruent
  (assert '(forall (?angle1 ?angle2)
               (iff (congruent ?angle1 ?angle2)
                    (= (measure ?angle1)
                        (measure ?angle2)))
   :name 'definition-of-congruency)

  ;; Parallel Lines
  (assert '(forall (?line1 ?line2)
               (iff (parallel ?line1 ?line2)
                    (and (not (intersects ?line1 ?line2))
                         (not (= ?line1 line2))))
   :name 'definition-of-parallel)

  (assert '(forall (?line1 ?line2)
               (iff (parallel ?line1 ?line2)
                    (parallel ?line2 ?line1)))
   :name 'parallel-lines-are-commutative)
;; Supplementary Angles
(assert '(forall (?angle1 ?angle2)
  (iff (supplementary ?angle1 ?angle2)
       (supplementary ?angle2 ?angle1))
  :name 'supplementary-is-commutative)
(assert '(forall (?angle1 ?angle2)
  (implies (supplementary ?angle1 ?angle2)
            (forall (?angle3)
             (iff (congruent ?angle1 ?angle3)
                  (supplementary ?angle2 ?angle3)))))

;; Vertical Angles
(assert '(forall (?angle1 ?angle2)
  (iff (vertical ?angle1 ?angle2)
       (vertical ?angle2 ?angle1))
  :name 'vertical-is-commutative)

;; Alternate Interior/Exterior Angles, Corresponding Angles
(dolist (fun '(AIA AEA CA))
  (assert '(forall (?angle1 ?angle2)
      (iff
       ,fun ?angle1 ?angle2
       ,fun ?angle2 ?angle1))
    :name 'aia-aea-ca-are-commutative)
  (assert '(forall (?angle1 ?angle2)
      (implies ,fun ?angle1 ?angle2)
      (congruent ?angle1 ?angle2))
    :name 'aia-aea-ca-are-congruent))

;; Hetch: H as in two parallel lines and a transversal
(assert '(forall ((p1 :sort line) (t1 :sort line) (p2 :sort line) (h :sort hetch))
  (implies (= h (hetch p1 t1 p2))
    (iff (and (parallel p1 p2)
              (intersects p1 t1)
              (intersects p2 t1))
         (and (AIA (a2 h) (a6 h))
              (AIA (a3 h) (a7 h))
              (AEA (a1 h) (a5 h))
              (AEA (a4 h) (a8 h))
              (CA (a1 h) (a3 h))
              (CA (a2 h) (a4 h))
              (CA (a8 h) (a6 h))
              (CA (a7 h) (a5 h))))))
)

(defun load-image-data ()
  (load "~/Desktop/image.txt"))

(defun load-text-data ()
  (assert '(parallel PS TV)))
(defun run-pos-test ()
  (setup-snark)
  (setup sorts)
  (setup-geometry)
  (load-image-data)
  (load-text-data)
  (prove '(supplementary SQW VUR)))

(defun run-neg-test ()
  (setup-snark)
  (setup sorts)
  (setup-geometry)
  (load-image-data)
  (load-text-data)
  (prove '(supplementary TUR VUW)))

(defun setup-load-data ()
  (setup-snark)
  (setup sorts)
  (setup-geometry)
  (load-image-data)
  (load-text-data))
LITERATURE CITED


