Solving Miller's Test of Intelligence using Propositional Representation
Project #1

Arvind Krishnaa Jagannathan
GT ID: 902891874

September 14, 2012

1 Introduction

1.1 Miller's Test for Intelligence

Miller's test for intelligence usually involves questions of the type A:B::C:x, where A,B,C are usually images which have some implicit relations. x is an unknown image, which needs to be selected from a set of given options, based on which of the choices best "fit" into the inferred relation [1]. The goal of this project is to be able to correctly solve three such problems, by representing each image as a propositional representation, and deriving the relation through this representation.

1.2 Propositional Representation

A propositional representation of any entity will consist of four components,

1. **Lexicon**: Describes the dictionary of valid terms which are used in the representation. Syntactically, it represents the vocabulary of the representation document.
2. **Structure**: It describes the relations that exists between the various terms/objects/entities in the representation.
3. **Procedure**: These are the actions or functions that can be performed on the representation, such as read/write values of entities. These may not make sense in the real world.
4. **Semantics**: These are the inferred relations that actually exist between the objects represented by the propositional representation.

1.2.1 A representation for the Miller's Test

This project has its own representational syntax for describing the images in each frame. Each question in this project has been consistently defined using the below described representation. The representations of each element of a particular question is stored in a text file. For eg., the propositional representation of question 1, is stored in "1-1.txt".

The various components of this representation (namely lexicon, structure, procedure and semantics) are described in detail in the following tables,

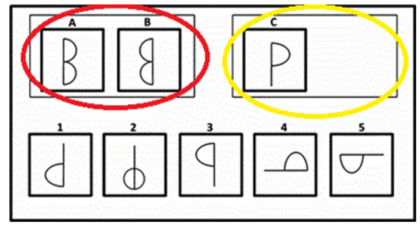
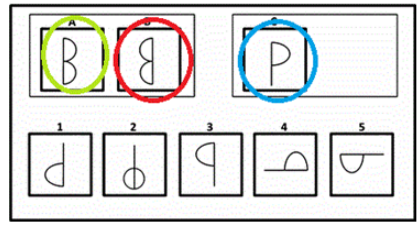
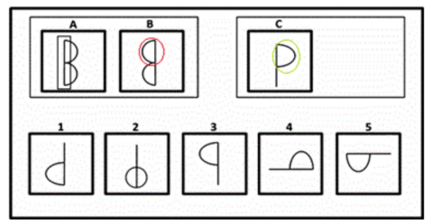
Lexicon	1. **Matrix:** A matrix is a container for a set of frames. In the Miller's test there are two matrices. The first is named a *"Reference"* matrix, as the relations among the frames in that matrix are taken as a reference to obtain the right choice. The second matrix is the *"Incomplete"* matrix. This implementation will try to fill in the empty frame with each answer choice, thereby leading to a *"Partially Complete"* matrix. If the relations that exist in that matrix match those in the reference matrix, then it becomes the *"Solution"* matrix.	
	2. **Frame:** A frame is a container for shapes, just like how a matrix is a container for frames. Each frame is characterized by a list of shapes present in them and a list of relations that exist between each of those shapes.	
	3. **Shape:** A shape in the context of this project is any object that is present inside a frame. A shape is characterized by its type and an identifier, although this is not necessary for this project. There can be infinitely many shapes within a frame and they can be of any type. As of now, the type attribute of each shape depends on the question provided, but it can be enhanced in the future, by making it an independent definition	

Figure 1: The various lexicons used in the representation

Structure	**Or in this case Frame-Relations:** These represent how the shapes within a frame are positioned spatially with respect to each other. This again depends on each problem, as the task of assigning a spatial position to a shape with respect to another is tightly reliant on what the "programmer" (in this case me) perceives is the right answer.	For instance, from frame A of the above image, the relations would be 1. Line1 "LeftOf" Semi-Circle1 2. Line1 "LeftOf" Semi-Circle2 3. Semi-Circle1 "TopOf" Semi-Circle2 and so on.

Figure 2: Syntax of how the representation's structure is defined

Procedure	The various procedures defined for this particular problem are: 1. Read representation: Read the propositional representation from the file of the entire problem, as well as for individual frames in the matrices of the image. 2. Parse representation: Once the file has been read, it needs to be parsed so that the attributes of each matrix, frame, shapes and relations in a frame can be deserialized and stored as Java objects for easy manipulation. 3. Build RelationMap: This procedure builds a mapping of a tuple<Shape1,Shape2,Relation> from one frame to a similar tuple of another frame. This data structure can then be analyzed to obtain some useful inferences between frames.	
Semantics	In this project, the semantics of the reference frame is obtained by analyzing the RelationMap datastructure described above. Once some useful inferences are made between frames A and B, a RelationMap is made for Frame C & Choice 1, Frame C & Choice 2, Frame C & Choice 3 and so on, each pair being a node in a semantic network. Inferences are derived for each node and the node for which the inferences exactly corelate with those of the reference is chosen as the answer.	For instance, the inferences drawn from the RelationMap of Frame A and Frame B (above) are: 1. Line1 "LeftOf" Semi-Circle1 ==> Line1 "RightOf" Semi-Circle1 2. Line1 "LeftOf" Semi-Circle2 ==> Line1 "RightOf" Semi-Circle2 3. Semi-Circle1 "TopOf" Semi-Circle2 ==> Semi-Circle1 "TopOf" Semi-Circle2 So the nodes which have the best co-relation of inferences compared to the above reference is Frame C & Choice1 and Frame C & Choice3, since 1. Line1 "LeftOf" Semi-Circle1 ==> Line1 "RightOf" Semi-Circle1 matches. In such cases, the Frame-Relations described in the propositional representation needs to be more specific (such as describing the vertical position of the semi-circles in each case)

Figure 3: Procedures and semantics inferred from the representation

2 Design of the Problem Solver

2.1 Algorithm

This is the general algorithm, which is used by the three questions to arrive at an answer. The elaboration of the algorithm below is applied to the propositional representations of each of the questions to determine the right choice.

2.1.1 Steps

1. First, the propositional representation describing the image in consideration is read to memory.
2. The above file is parsed by the SIMPLEPROPOSITIONALREPRESENTATIONREPRESENTATIONPARSER which then stores the information into corresponding objects. For example, the details of each SHAPE is stored into a corresponding instance of a SHAPE class. Each FRAME object consists of a list of SHAPES and a list of FRAME-RELATIONS.
FRAME-RELATIONS is defined as a triplet of

$$\langle Shape1\rangle\langle Shape2\rangle\langle Relationship\rangle$$

where *Relationship* is the relative positioning of *Shape1* with respect to *Shape2*.

3. Once the data is deserialized to objects, the learning and reasoning part of the program begins.

4. Group the frames A and B as the reference matrix.

5. Store all the relationships among the shapes in frame A as a list (RELATIONLIST1). Similarly construct such a list for the frame B (RELATIONLIST2)

6. Traverse through RELATIONLIST1, obtaining the corresponding $\langle Shape1\rangle and\langle Shape2\rangle$.
  6.1 For each ordered pair $\langle Shape1\rangle, \langle Shape2\rangle$, iterate through RELATIONLIST2 to check if there is a relation with the same ordering of shapes.
For example, if there is a relation in RELATIONLIST1 like: CIRCLE1 "LEFTOF" CIRCLE2, then check in RELATIONLIST2 if there is a relation where $\langle Shape1\rangle$ = CIRCLE1 and $\langle Shape2\rangle$ = CIRCLE2
  6.2 If such a relation is found, then add it (the first instance!) to a *HashMap*, where the key is the FRAME-RELATIONS triplet from RELATIONLIST1 and the value is the $\langle Relationship\rangle$ from RELATIONLIST2. This is called the relationMap, and this step constitutes the learning phase.
  6.3 Do not consider the relation from RELATIONLIST2 for further steps.
NOTE: Step 6 has gathered the implicit relationships between frames A and B. The relationMap is just a description of how each shape has "transformed" with respect to each other from A to B.

7. Also store the following to memory,
  7.1 Number of shapes in frame A - Number of shapes in frame B = DIFFSHAPES
  7.2 Number of relations in frame A - Number of relations in frame B = DIFFRELATIONS

8. Repeat the following steps for every answer choice,
  8.1 Group frame C and the current frame of the answer choice (say frame I) into a matrix (The "Partially Complete" matrix).
  8.2 Perform steps 5 to 7 for this matrix, where frame C takes the position of frame A and frame I takes the position of frame B.
  8.3 *Quick Prune Step*:
  Check if DIFFSHAPES(Partially Complete Matrix) = DIFFSHAPES(Reference Matrix).
  Check if DIFFRELATIONS(Partially Complete Matrix) = DIFFRELATIONS(Reference Matrix).

If either of the above two tests fail, that means the answer frame in consideration , frame I, does not correspond to frame C, in the same way as frame B corresponds to frame A [1]. Ignore this frame and continue with the next frame among the answer choices .

8.4 Otherwise, find the correlation of the *relationMap* of the "Partially Complete" matrix to the *relationMap* of the "Reference" matrix. To do this, perform the following steps

(a) Check if a key in the *relationMap* of the "Partially Complete" matrix is present in the *relationMap* of the "Reference" matrix.

(b) If yes, check if their values are the same. If the values are not the same, the frame in consideration is not the right answer and ignore it.

8.5 Repeat the above step till all the keys of the *relationMap* of the "Partially Complete" matrix are exhausted. If the frame under consideration has passed check 8.4(a) and 8.4(b), then this is right answer (since the "implicit" relation(s) between frame C and frame I, are exactly matching the "implicit" relation(s) between frame A and frame B). Return the frame identifier of this frame.

9. If no frame identifier has been returned from step 8, then print an *Error* message!

2.1.2 Pseudocode

This subsection lists out the most important aspects of the algorithm as pseudocode. Trivial functions used in these major functions are not listed out.

The *AnalogSolver* function is the **"Reasoning"** part of the intelligent agent, which is a driver to the actual reasoning function *IsMatching*
The *BuildRelationMap* function is the **"Learning"** part of the intelligent agent, which is used to create the semantic network among the inferences.

Algorithm 1 Analogy Solver Agent

```
function AnalogySolver(PropositionalRepresentation)
    (ReferenceMatrix) ← Parse(PropositionalRepresentation)
    (IncompleteMatrix) ← Parse(PropositionalRepresentation)
    for all frameI : answerChoiceFrames do
        (PartiallyCompleteMatrix) ← (IncompleteMatrix) + frameI
        (diffShapes1, diffRelations1) ← GetDiff(ReferenceMatrix)
        (diffShapes2, diffRelations2) ← GetDiff(PartiallyCompleteMatrix)
        if diffShapes1! = diffShapes2ORdiffRealtions1! = diffRelations2 then
            discard(frameI)                                    ▷ Quick Prune Step
            continue;
        else
            relationMap1 ← BuildRelationMap(ReferenceMatrix)
            relationMap2 ← BuildRelationMap(PartiallyCompleteMatrix)
            match = isMatching(relationMap1, relationMap2)
            if match = TRUE then
                return Identifier(frameI)
            end if
        end if
    end for
end function
```

[1] This rule is based on my assumption that the variation of the shapes and therefore their corresponding relations will be in the same ratio in the "Solution" matrix as in the "Reference" matrix. It is possible that this rule may not be true, but intuition suggests otherwise

Algorithm 2 Checking the correlation of the relationMaps

```
function isMatching(relationMap1, relationMap2)
    for all relation : relationMap2 do
        key2 ← GetKey(relation)
        for all (keyList1 = GetKey(relation)) : relationMap1 do
            if Contains(keyList1, key2) then
                if GetValue(relationMap1, key2)! = GetValue(realtionMap2, key2) then
                    return FALSE
                end if
            end if
        end for
        return TRUE
    end for
end function
```

Algorithm 3 Building the relation map for a given matrix

```
function BuildRelationMap(matrix)
    (framesList) ← GetFrames(matrix)                    ▷ Two-element list
    frame1 ← Get(framesList, 0)
    frame2 ← Get(framesList, 1)
    for all relation1 : GetRelations(frame1) do
        for all relation2 : GetRelations(frame2) do
            if Shape1(relation2) = Shape1(relation1)ANDShape2(relation2) = Shape2(relation1) then
                relationMap = AddToMap(relation1, Relationship(relation2))
            end if
        end for
    end for
    return relationMap
end function
```

2.2 Architecture

The architecture diagram of the Miller's Analogy solver is depicted in Figure 4. There are 3 main components in the architecture, which correspond to the three functions described above as pseudocode.

1. RelationMap Builder: Builds the relationMap for each of the matrices. (Matrix 1= FrameA + FrameB & Matrix 2 = FrameC + Frame I)
2. Correlation Checker: Checks for the match in the correlation of the two relationMaps, provided Matrix 2 has passed the prune check.
3. Output: Which is the value returned by the *AnalogySolver*

3 Implementation

The entire project has been implemented in Java. The GUI part of the project was built using Netbeans IDE. The entire lexicon of the propositional representation is mapped onto corresponding user-defined Java classes of the same name. The class diagram in Figure 5 gives an overview of how the program is structured. It does not list the classes used in the GUI implementation for the purpose of readability. Also since the only class used in the GUI is Swing's Frame class, including that in the class diagram would not make sense.

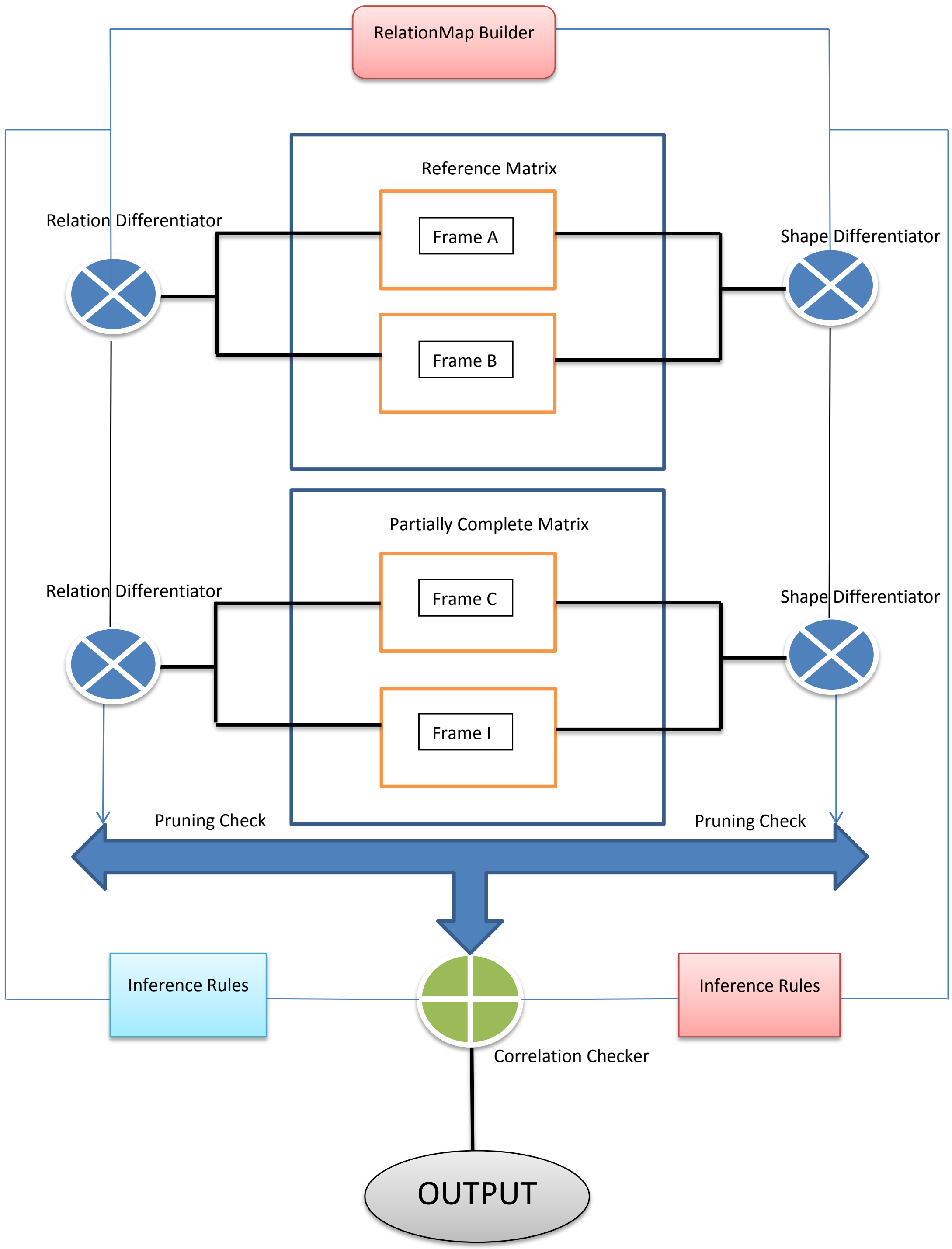


Figure 4: Architecture of the agent

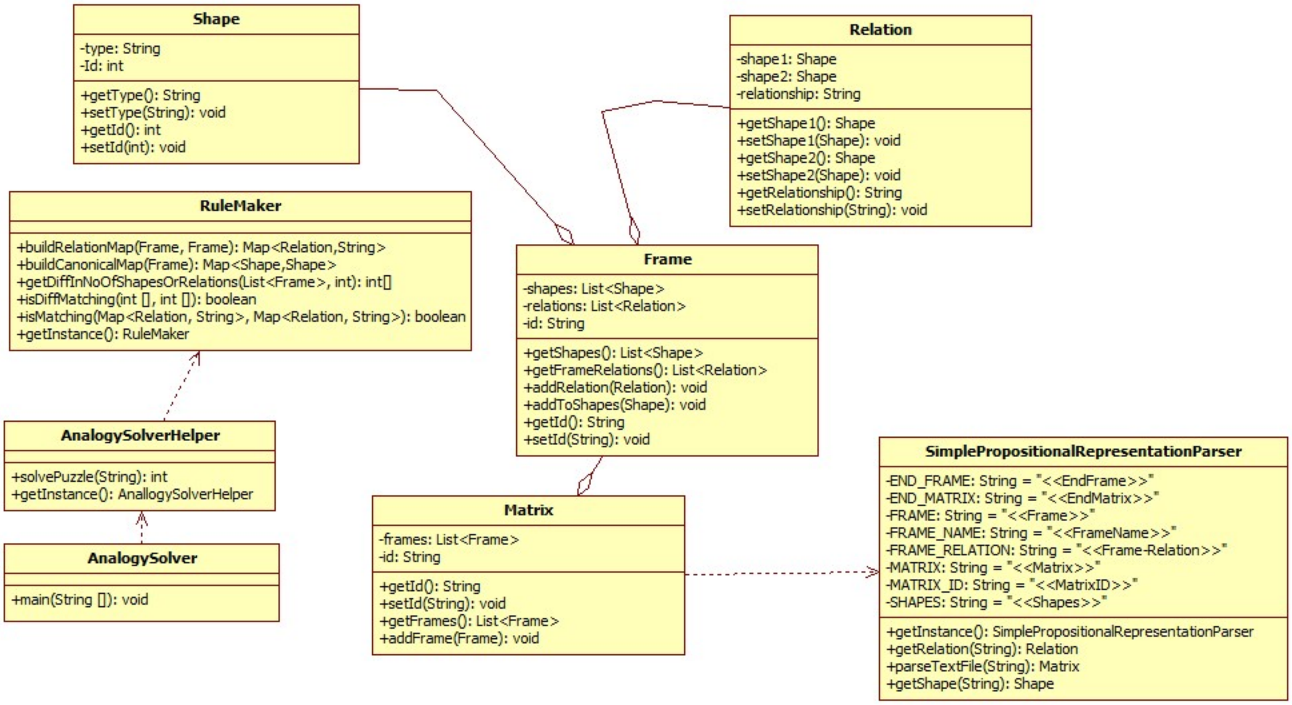


Figure 5: Java Class Diagram

4 Introspection

4.1 Evaluation of the algorithm

The algorithm developed to solve Miller's test is quite robust and models the human thought process greatly. This algorithm can extended wth some effort to solve problems where each matrix may have more than two frames. Most of the core functions already support multiple frames within a matrix, except the *buildRelationMap()* method.

Also, since the method does not restrict itself to conventional shapes, any shape can be part of the image of the question and it would still be possible to represent it consistently.

4.1.1 Time and Space Complexity

Time Complexity: The most time consuming part of this approach is to find the co-relation among the *relationMap*s of two matrices. Since this involves two for-loops (each with a constant lookup as it occurs on a Map), it would have a time complexity of $\mathcal{O}n^2$. Also this checker would need the output of the *buildRelationMap()*, which also iterates over two for-loops (having a complexity also $\mathcal{O}n^2$). Hence the actual time-complexity for the correlation checker is $\mathcal{O}n^4$. However the program runs significantly better than $\langle NumberOfChoices \rangle * n^4$, because of the pruning of some of the answer choices prior to the correlation check.

Space Complexity: The program requires to store in memory a *HashMap* of the relations, a list of Matrices, which in turn will have a list of Shapes, Relations etc., However, none of these have more than a linear requirement, and so the space complexity would be $\mathcal{O}n$.

When the problem expands, the time as well as space complexity would also increase linearly, since they both depend on the number of frames and the number of shapes and relations in each frame.

4.1.2 Positives of the algorithm

- Models how most humans think while approaching a problem of the type A:B::C:x
- Offers a fair amount of generality, as there is no restriction on the number of frames, the type of shapes, its orientation and so on.
- Given a strict propositional representation, this algorithm should intuitively always give a correct answer.
- The learning and the reasoning phase can be clearly seen as the program executes.

4.1.3 Criticisms

- The algorithm is untested when the answer choices are rogue. For instance, if one of the answer choices itself were a blank image, then the algorithm would always choose that. Also it is entirely possible that the algorithm says that none of the choices are part of the solution, whereas in reality there might be some implicit relationship which was too subtle for the algorithm to record.
- The quick-prune step is incorporated purely out of intuition. There is no way to prove if its a valid assumption or not.
- Since the algorithm expects the propositional representation to be very strict, we need to list out the relations among shapes in a frame exhaustively. This might sometime lead to double relations like *(A "LeftOf" B, B "RightOf" A)*, which may have been avoided. Thus a flexible representational schema is not possible as of now.
- Since the representation is in a text file, the parsing logic may not be the most efficient. Even though the parsing logic runs in $\mathcal{O}n$ time as it traverses the file line-by-line, overheads exist in performing *if-checks*. Instead if some structured format was used for input, such as XML, parsing would have been most efficient, as Java and many other languages have their own parsers for XML.
- The logging on the console in this project is primarily done using *System.out*, which is not a good Java practice. For future projects, some logging mechanism, like Java's *SimpleLogger* [2] or Apache's *Log4j* [3] will be used.

References

[1] Patrick Henry Winston, "Artificial Intelligence", Third Edition, Addison-Wesley Publishing Company, 1992.

[2] Java online documentation, http://www.oracle.com/technetwork/java/javase/downloads/index.html

[3] Apache Log4J, http://logging.apache.org/log4j/1.2/