AI agent with meta cognition

[Arti Chauhan: Dec-05 2016]

Introduction

This paper describes design of an AI agent that can not only solve Raven's Progressive Matrices (RPM) problem but also has meta-cognition capability to generate explanation about its problem solving approach.
My P3 Agent reasons over images at high level of abstraction, without any shape detection. Though this technique worked quite nicely for P3, it doesn't extend itself very well to answer some of the questions asked in Final exam. To that end, I propose to augment agent's design where information extracted from images in deliberative layer is stored in knowledge structure (Frames), which facilitates in building an explanation by metacognition layer.

Requirement

Agent should be able to take questions from user and be able to provide a satisfactory explanation for it. At very minimum, it should be able to answer following questions.

1. *What is the object in the eight images displayed in the matrix?*
2. *What is the relationship between the three images in the top row of the matrix?*
3. *What is the relationship between the three images in the middle column?*
4. *What is the difference between the #3 and #4 as potential answers?*
5. *Why did you select #5 as the answer?*
6. *Why is #3 not the correct answer?*

Design

To satisfy above requirement, Agent employs 3 main tasks, which are described in detail in subsequent section.

1. *Understand user's question.*
2. *Create explanation for question asked.*
3. *Output response in a human-friendly manner.*

Figure below shows approximate mapping between above-mentioned tasks, Agent's functional modules, AI architecture and KBAI concepts used in this design.

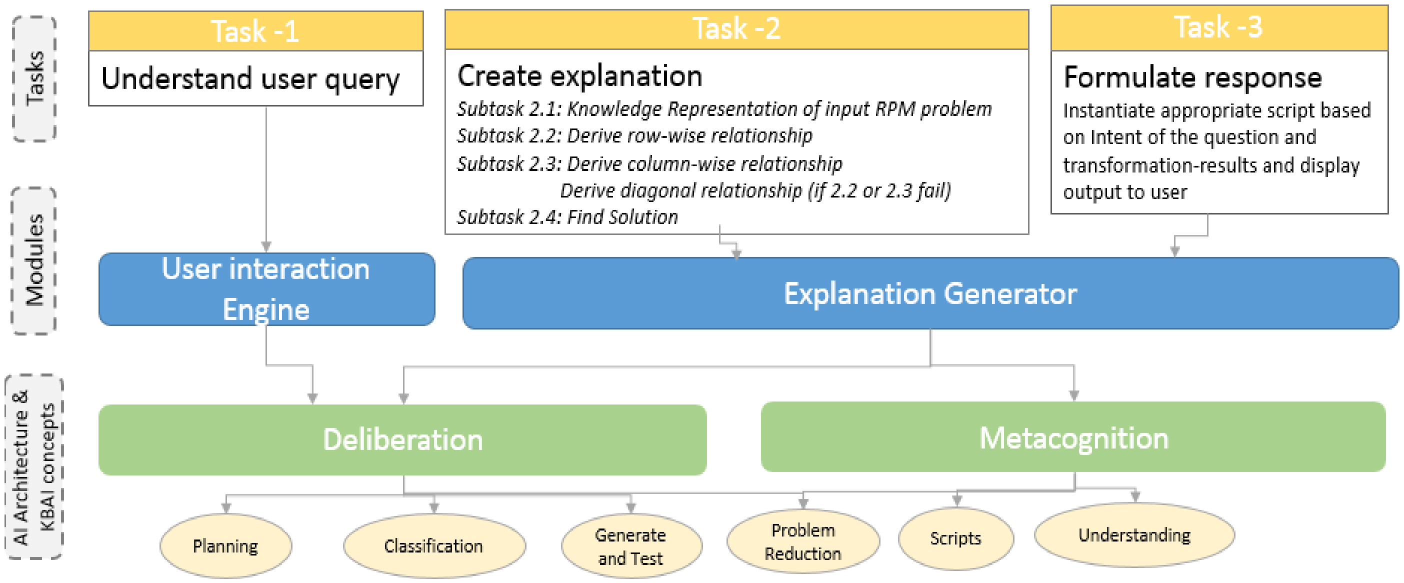
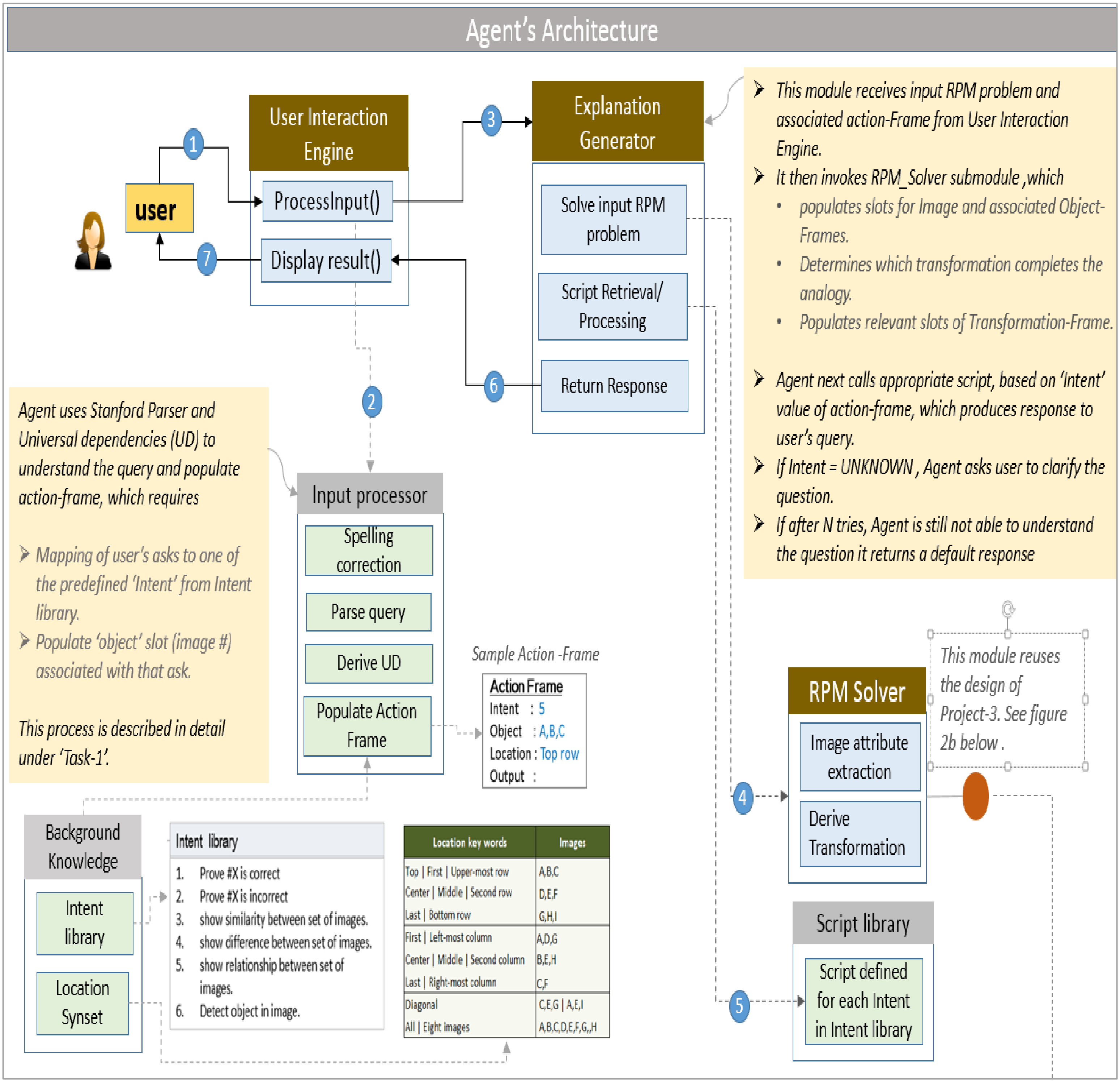

Tasks
Task -1
Understand user query
Task -2
Create explanation
Subtask 2.1: Knowledge Representation of input RPM problem
Subtask 2.2: Derive row-wise relationship
Subtask 2.3: Derive column-wise relationship
Derive diagonal relationship (if 2.2 or 2.3 fail)
Subtask 2.4: Find Solution
Task -3
Formulate response
Instantiate appropriate script based on Intent of the question and transformation-results and display output to user
Modules
User interaction Engine
Explanation Generator
AI Architecture & KBAI concepts
Deliberation
Metacognition
Planning
Classification
Generate and Test
Problem Reduction
Scripts
Understanding

Figure: 1



- This module receives input RPM problem and associated action-Frame from User Interaction Engine.
- It then invokes RPM_Solver submodule ,which
  - populates slots for Image and associated Object-Frames.
  - Determines which transformation completes the analogy.
  - Populates relevant slots of Transformation-Frame.
- Agent next calls appropriate script, based on 'Intent' value of action-frame, which produces response to user's query.
- If Intent = UNKNOWN , Agent asks user to clarify the question.
- If after N tries, Agent is still not able to understand the question it returns a default response

Agent uses Stanford Parser and Universal dependencies (UD) to understand the query and populate action-frame, which requires

- Mapping of user's asks to one of the predefined 'Intent' from Intent library.
- Populate 'object' slot (image #) associated with that ask.

This process is described in detail under 'Task-1'.

Sample Action -Frame

Action Frame
Intent : 5
Object : A,B,C
Location : Top row
Output :

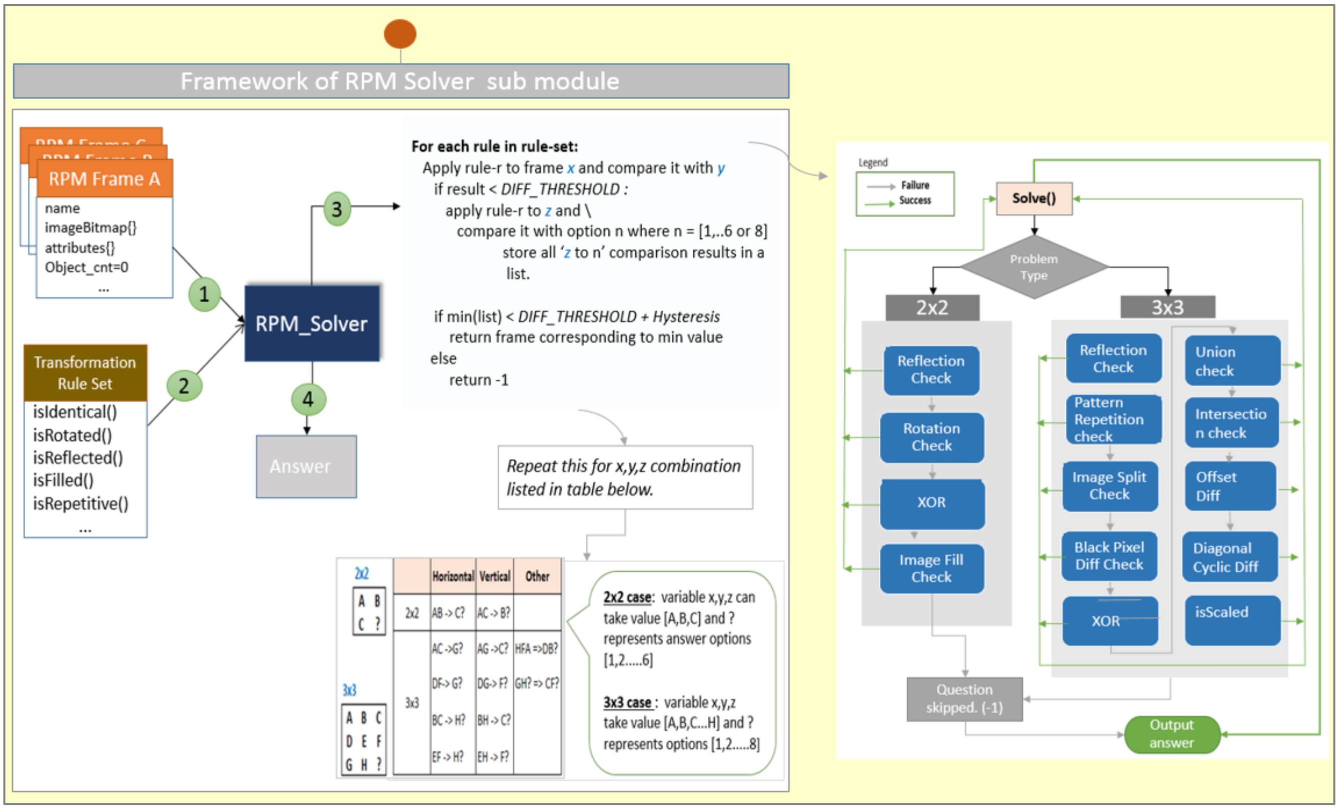
Intent library

1. Prove #X is correct
2. Prove #X is incorrect
3. show similarity between set of images.
4. show difference between set of images.
5. show relationship between set of images.
6. Detect object in image.

Location key words	Images
Top | First | Upper-most row	A,B,C
Center | Middle | Second row	D,E,F
Last | Bottom row	G,H,I
First | Left-most column	A,D,G
Center | Middle | Second column	B,E,H
Last | Right-most column	C,F
Diagonal	C,E,G | A,E,I
All | Eight images	A,B,C,D,E,F,G,,H

This module reuses the design of Project-3. See figure 2b below .

Figure: 2a



```
For each rule in rule-set:
  Apply rule-r to frame x and compare it with y
    if result < DIFF_THRESHOLD :
      apply rule-r to z and \
        compare it with option n where n = [1,..6 or 8]
                  store all 'z to n' comparison results in a
                  list.

    if min(list) < DIFF_THRESHOLD + Hysteresis
      return frame corresponding to min value
  else
      return -1
```

2x2			
A	B		
C	?		

3x3		
A	B	C
D	E	F
G	H	?

	Horizontal	Vertical	Other
2x2	AB -> C?	AC -> B?	
3x3	AC ->G?	AG ->C?	HFA =>DB?
	DF-> G?	DG-> F?	GH? => CF?
	BC -> H?	BH -> C?	
	EF -> H?	EH -> F?	

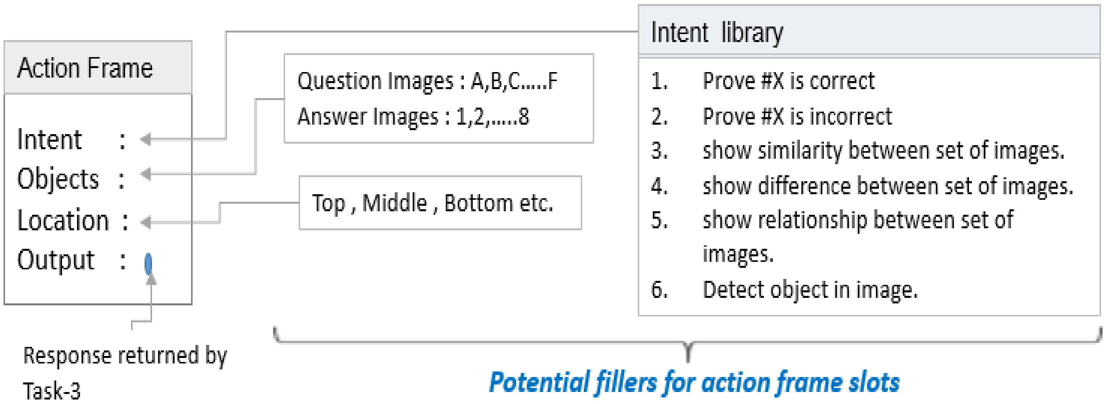
2x2 case: variable x,y,z can take value [A,B,C] and ? represents answer options [1,2.....6]

3x3 case : variable x,y,z take value [A,B,C...H] and ? represents options [1,2.....8]

Figure: 2b

Task 1: Understanding user's input/question

Agent is preconfigured with a library of 'Intents'. Goal of this task is to interpret the question and determine appropriate 'Intent' slot in action frame and objects (ie images) associated with that Intent.


Action Frame
Intent :
Objects :
Location :
Output :
Response returned by Task-3
Question Images : A,B,C.....F
Answer Images : 1,2,.....8
Top , Middle , Bottom etc.
Intent library
Potential fillers for action frame slots

1. Prove #X is correct
2. Prove #X is incorrect
3. show similarity between set of images.
4. show difference between set of images.
5. show relationship between set of images.
6. Detect object in image.

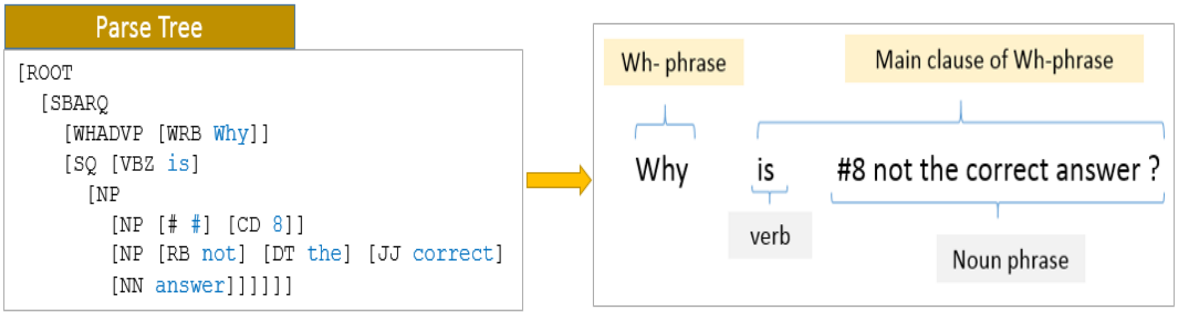
Figure: 3

In order to make sense of the question posed to it, Agent has to be equipped with rules of English grammar and part of speech, which will help Agent derive semantic understanding of the question/sentence. Agent will use Stanford parser and universal dependencies (UDs) for this purpose. I will explain this process with help of two examples, which in essence covers interpretation of most of the questions mentioned in Final-Exam.

Example-1 Why is #8 not the correct answer ?

Agent will parse this sentence and derive UDs as shown below in Figure-3a and 3b.

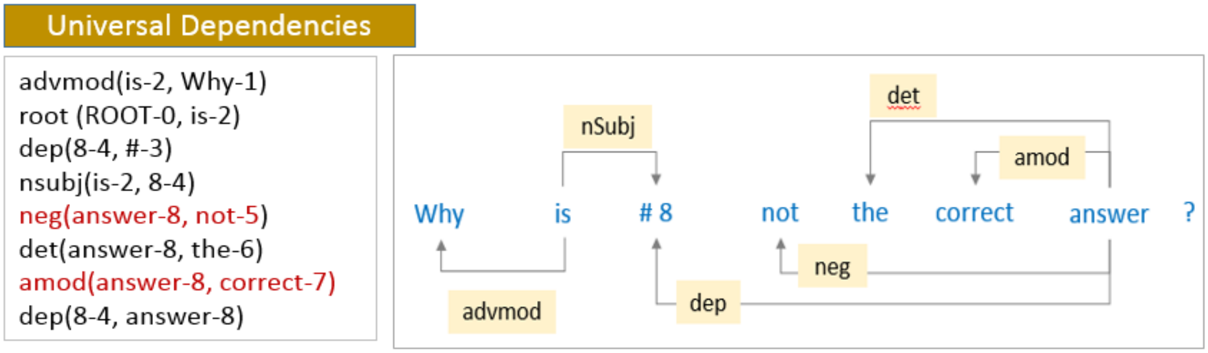
- *SBARQ tag in Parse-Tree helps Agent understand that user input is a question, introduced by Wh-phrase.*
- *SQ ("is #8 not the correct answer") is the main clause of Wh-question. Verb 'is' is followed by a noun phrase.*
- *From dependency tree (Figure-3b), Agent determines subject is a cardinal number '8'. So it knows that '#8' is the image question being asked about and assigns this to 'object' slot in action frame.*
- *In order to understand what exactly about image-8 is being asked, it has to further analyze the noun phrase "not the correct answer". Figure-3b also tells agent that subject '#8' has dependency on noun 'answer' which has a negation ('not') and an adjective 'correct' associated with it.*



Parse Tree

```
[ROOT
  [SBARQ
    [WHADVP [WRB Why]]
    [SQ [VBZ is]
      [NP
        [NP [# #] [CD 8]]
        [NP [RB not] [DT the] [JJ correct]
        [NN answer]]]]]]
```

Figure: 3a

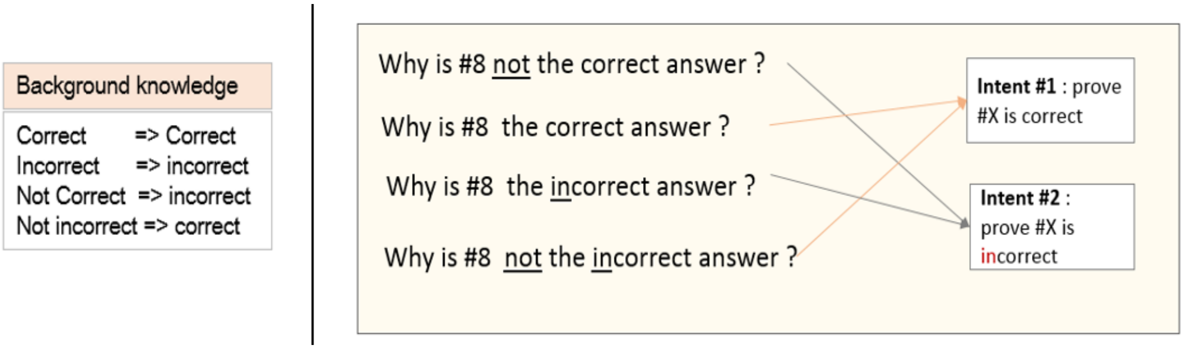


Universal Dependencies

advmod(is-2, Why-1)
root (ROOT-0, is-2)
dep(8-4, #-3)
nsubj(is-2, 8-4)
neg(answer-8, not-5)
det(answer-8, the-6)
amod(answer-8, correct-7)
dep(8-4, answer-8)

Figure: 3b

- *From its background knowledge, Agent knows that word 'not' and 'correct' maps to 'incorrect' and hence maps to 'Intent' slot to #2 ('Prove #x is incorrect')*
- *Note that this background knowledge also helps agent make sense of negation of negation. Let's say user poses a question 'why is #8 not the incorrect answer?' Agent learns negation 'not' and adjective 'incorrect' associated with subject (#8). It checks it background knowledge which tells it that 'Not Incorrect' => correct, which helps agent map this question to right Intent (#1)*

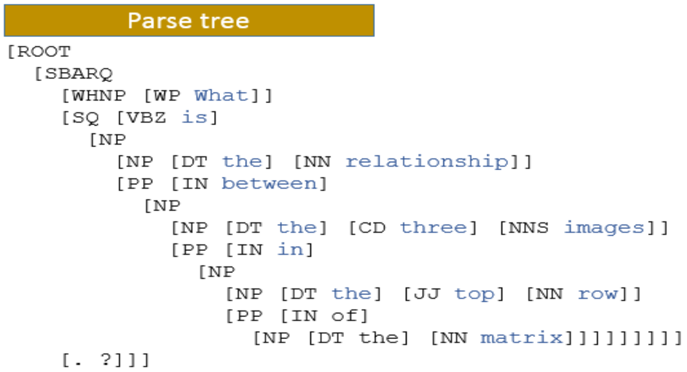


Background knowledge	
Correct	=> Correct
Incorrect	=> incorrect
Not Correct	=> incorrect
Not incorrect	=> correct

Why is #8 not the correct answer ?
Why is #8 the correct answer ?
Why is #8 the incorrect answer ?
Why is #8 not the incorrect answer ?
Intent #1 : prove #X is correct
Intent #2 : prove #X is incorrect

Figure: 3c

Example-2 What is the relationship between the three images in the top row of the matrix?



Parse tree

```
[ROOT
  [SBARQ
    [WHNP [WP What]]
    [SQ [VBZ is]
      [NP
        [NP [DT the] [NN relationship]]
        [PP [IN between]
          [NP
            [NP [DT the] [CD three] [NNS images]]
            [PP [IN in]
              [NP
                [NP [DT the] [JJ top] [NN row]]
                [PP [IN of]
                  [NP [DT the] [NN matrix]]]]]]]]]
    [. ?]]]
```

Figure: 4a

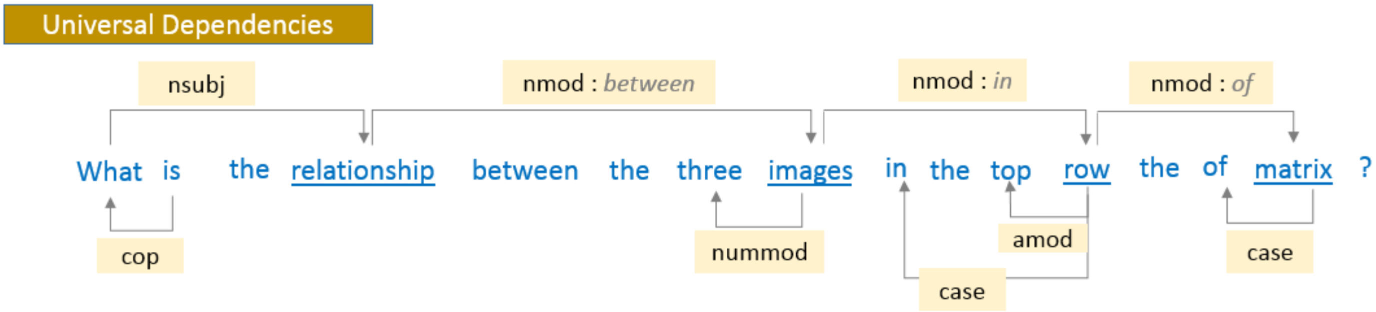

Universal Dependencies
nsubj
nmod : between
nmod : in
nmod : of
What is the relationship between the three images in the top row the of matrix ?
cop
nummod
amod
case
case

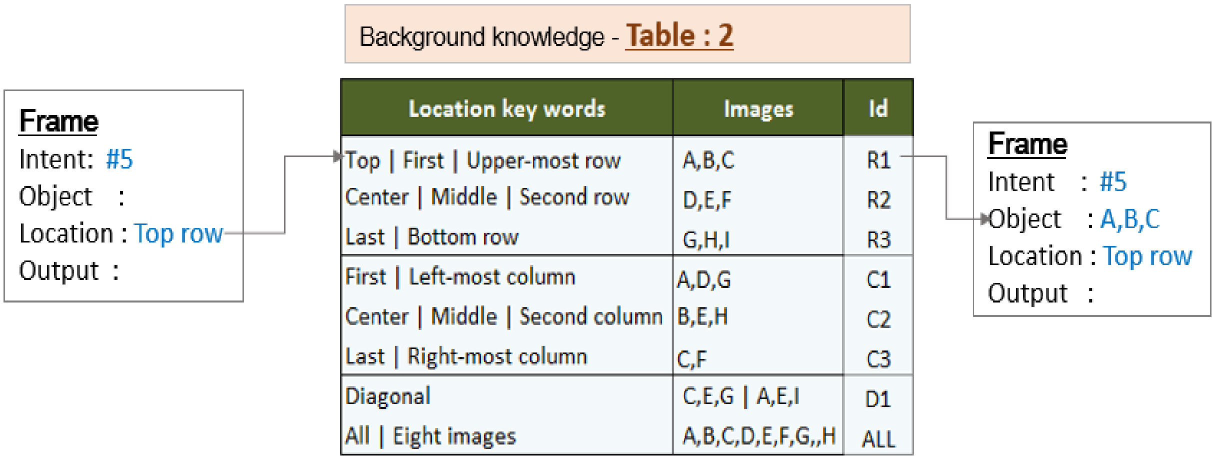
Figure: 4b

Agent proceeds with same procedure as described in previous example. After parsing and retrieving UDs, it can understand that this a question (SBARQ tag) where copula 'what is' has subject 'relationship' associated with it. This information helps Agent fill slot 'Intent' with #5 ("*show relationship between set of images*.")

It further analyses the noun phrase ('*the three images in the top row of the matrix*') to determine relationship between which images is being asked about.

1. *From dependency tree (Fig-4b), Agent learns that 'relationship' between 'images' of 'row' of a 'matrix' is being asked about.*
2. *'Top' (Adjective Modifier - amod) for noun 'row' tells agent which row is being asked about. In order to fill 'object' slot, Agent probes its knowledge to determine which images map to 'Top row'. In this case, memory returns images A, B, C (Fig-4c)*
3. *Note that Agent is supplied with knowledge (Table-2) that acts as a mini Synset and helps Agent understand that word 'top' , 'First', Upper-most' - all mean the same and hence resolve to image set A,B,C.*

Once Agent has interpreted user's input and successfully filled 'Intent' and 'Object' slots, it moves to Task-2 to create explanation.



Background knowledge - Table : 2

Location key words	Images	Id
Top | First | Upper-most row	A,B,C	R1
Center | Middle | Second row	D,E,F	R2
Last | Bottom row	G,H,I	R3
First | Left-most column	A,D,G	C1
Center | Middle | Second column	B,E,H	C2
Last | Right-most column	C,F	C3
Diagonal	C,E,G | A,E,I	D1
All | Eight images	A,B,C,D,E,F,G,,H	ALL

Figure: 4c

Task 2: Creating explanation

This task relies heavily on Agent's deliberative layer which is responsible for extracting visual representation of images and reasons over it. Agent uses **'Classification'** to categorize RPM problems either as 2x2 or 3x3 and then invokes appropriate plan to solve that problem. Each **Plan** has a rule-set that helps agent determine sequence of transformation to be checked and populates appropriate slots in Agent's RPM knowledge construct (Figure-5a). Agent uses **Frames** for knowledge representation and **Generate and Test** in conjunction with planning for problem solving. With help of Pillow (Python Image Processing) library, Agent employs **Visuospatial Analogy** at its core to solve the problem. This aspect of Agent's design is described in detail in P-3 Reflection. Here' I will cover just the overview. In summary, Agent executes following steps to derive the answer and gathers all the necessary information to be able to explain itself.

1. *Initialize and populate Image-Frame and Object-Frame data structure (RPM Knowledge construct) when new RPM problem is received.* *[Subtask 2.1]*
2. *Derive row-wise relationship.* *[Subtask 2.2]*
3. *Derive column-wise relationship.* *[Subtask 2.3]*
    - *Derive diagonal relationship if #2 or #3 don't provide an answer within confidence threshold.*
4. *Find solution image that satisfies #2 and #3.* *[Subtask 2.4]*

Below is walk-through of above-mentioned fours steps for example problem from Final-Exam.

Subtask 2.1: Knowledge Representation of input RPM problem

Agent receives a RPM problem and extracts attributes (such as width and length of bounding box, number of objects) for each image and attributes of each object (shape, length, width, area, center etc.) within that image and populates slots for Image-Frame and corresponding Object-Frame(s) as shown in Figure-5a.

Note that only default slots for Image and Object Frames are shown below. However, Agent can dynamically insert new slots (Eg: angle or BW_pixel_ratio) depending on type of transformation being tested.

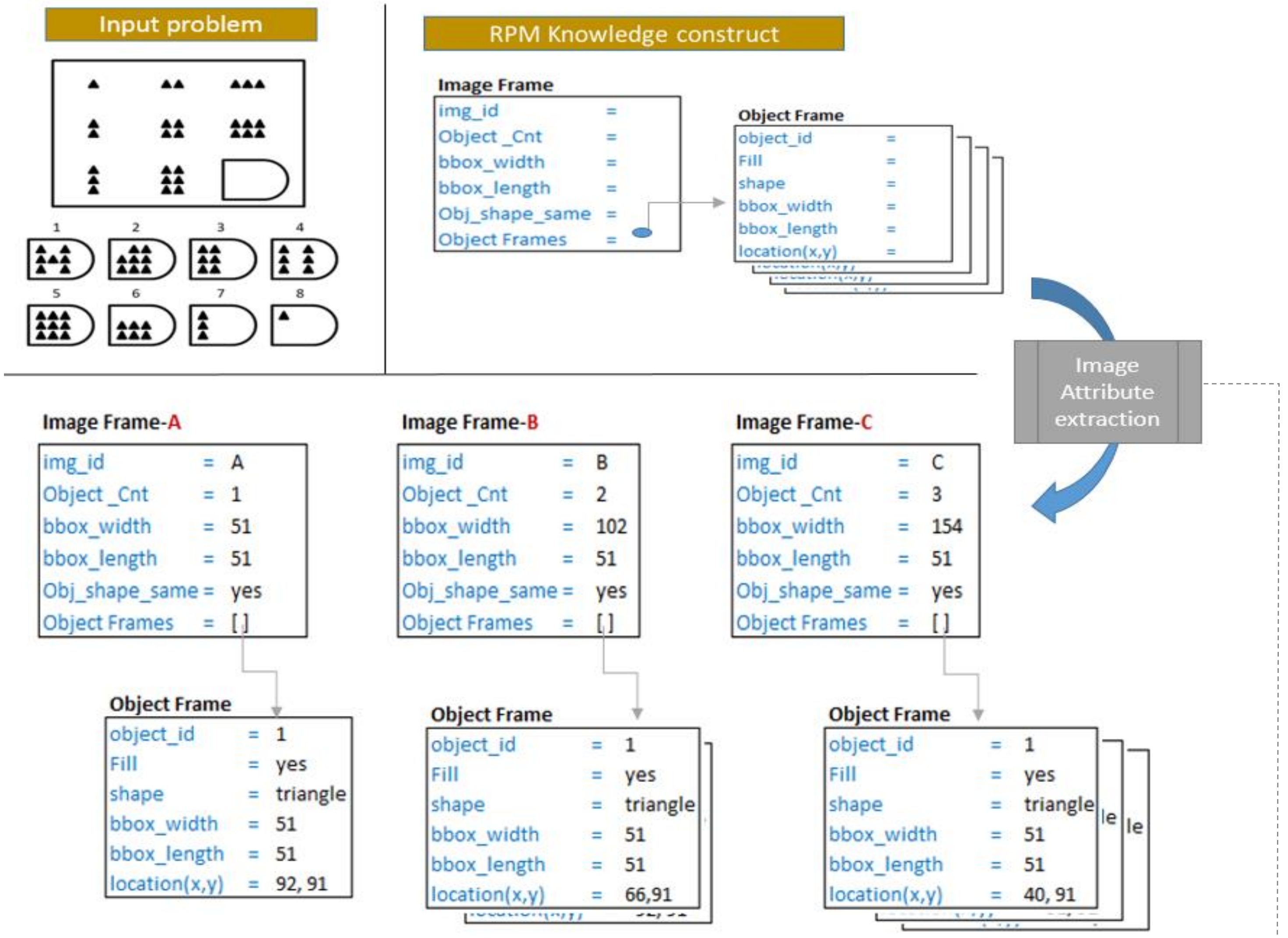


Image Frame

img_id =
Object _Cnt =
bbox_width =
bbox_length =
Obj_shape_same =
Object Frames =

Object Frame

object_id =
Fill =
shape =
bbox_width =
bbox_length =
location(x,y) =

Image Frame-A

img_id = A
Object _Cnt = 1
bbox_width = 51
bbox_length = 51
Obj_shape_same = yes
Object Frames = [ ]

Image Frame-B

img_id = B
Object _Cnt = 2
bbox_width = 102
bbox_length = 51
Obj_shape_same = yes
Object Frames = [ ]

Image Frame-C

img_id = C
Object _Cnt = 3
bbox_width = 154
bbox_length = 51
Obj_shape_same = yes
Object Frames = [ ]

Object Frame

object_id = 1
Fill = yes
shape = triangle
bbox_width = 51
bbox_length = 51
location(x,y) = 92, 91

Object Frame

object_id = 1
Fill = yes
shape = triangle
bbox_width = 51
bbox_length = 51
location(x,y) = 66,91

Object Frame

object_id = 1
Fill = yes
shape = triangle
bbox_width = 51
bbox_length = 51
location(x,y) = 40, 91

Figure: 5a

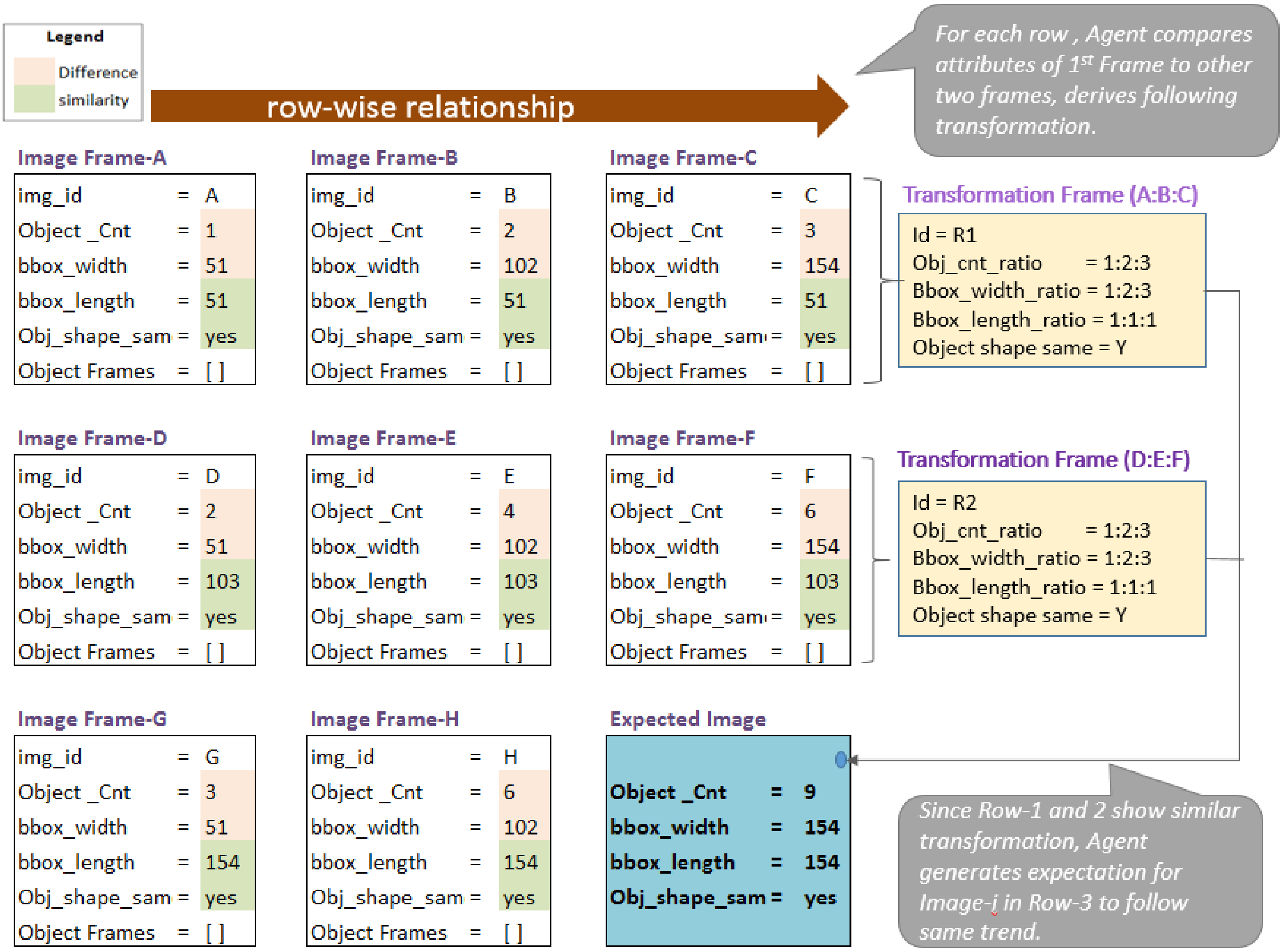
Brief description of processing inside 'Image Attribute Extraction' module in Fig-5a

- **Object extraction/count**: Agent employs Connected Component (CC) analysis for this purpose. It uses Depth First Search to compute number of objects in the image and stores black pixel location of each CC (object) and total black pixels in a python dictionary.
- **Bbox_width /length**: Uses Pillow's bounding box function (W= (x1-x0) L = (y1-y0))
- **Location**: compute x, y coordinates of center of the object (x0+W/2, y0+L/2)
- **Fill:** Total black pixels in the object / black pixels on perimeter .Agents creates a new image that displays only the edge of the object (ImageFilter.FIND_EDGES) and compares it with original object-image. Ratio of >95% - Fill , <5% - Not Fill else partially filled)
- **Shape:** Agent first computes area by summing black pixels in perimeter and internal white pixels. For this purpose Agent uses new image that it created via (ImageFilter.FIND_EDGES). It compares object's area with area of known images (square, rhombus, circle, triangle, hexagon, and octagon).If this comparison is not within a confidence threshold, object is classified as 'unknown'.
- **Object pairing -** This is required when set of images being compared has more than one object.

Subtask 2.2: Deriving row-wise relationship

Agent compares attribute of each image in a row and captures which attributes changed (and to what degree) and which remained same. In example below, attributes that remained same across the row are marked green and the ones that changed are in red. Agent captures this variances in Transformation-Frame (shown on the right). Eg: object count changed from 1->2->3 as we move from A->B->C but shape of the object(s) remains the same.

[Note: While deriving row/column/diagonal relationship, Agent compares both Image and Object Frame attributes. Since object-Frame attributes don't change from one image to another in this example, Object-Frames are not shown in Figure 5a-5d for the sake of readability]



For each row , Agent compares attributes of 1st Frame to other two frames, derives following transformation.

Image Frame-A

img_id	=	A
Object _Cnt	=	1
bbox_width	=	51
bbox_length	=	51
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-B

img_id	=	B
Object _Cnt	=	2
bbox_width	=	102
bbox_length	=	51
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-C

img_id	=	C
Object _Cnt	=	3
bbox_width	=	154
bbox_length	=	51
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Transformation Frame (A:B:C)

Id = R1
Obj_cnt_ratio = 1:2:3
Bbox_width_ratio = 1:2:3
Bbox_length_ratio = 1:1:1
Object shape same = Y

Image Frame-D

img_id	=	D
Object _Cnt	=	2
bbox_width	=	51
bbox_length	=	103
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-E

img_id	=	E
Object _Cnt	=	4
bbox_width	=	102
bbox_length	=	103
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-F

img_id	=	F
Object _Cnt	=	6
bbox_width	=	154
bbox_length	=	103
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Transformation Frame (D:E:F)

Id = R2
Obj_cnt_ratio = 1:2:3
Bbox_width_ratio = 1:2:3
Bbox_length_ratio = 1:1:1
Object shape same = Y

Image Frame-G

img_id	=	G
Object _Cnt	=	3
bbox_width	=	51
bbox_length	=	154
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-H

img_id	=	H
Object _Cnt	=	6
bbox_width	=	102
bbox_length	=	154
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Expected Image

Object _Cnt	**=**	**9**
bbox_width	**=**	**154**
bbox_length	**=**	**154**
Obj_shape_sam	**=**	**yes**

Since Row-1 and 2 show similar transformation, Agent generates expectation for Image-i in Row-3 to follow same trend.

Figure: 5b

Subtask 2.3: Deriving column-wise relationship

Same as subtask 2.2 except processing now occurs column-wise and variance is captured in Transformation-Frame as shown at the bottom of Figure-5c below.

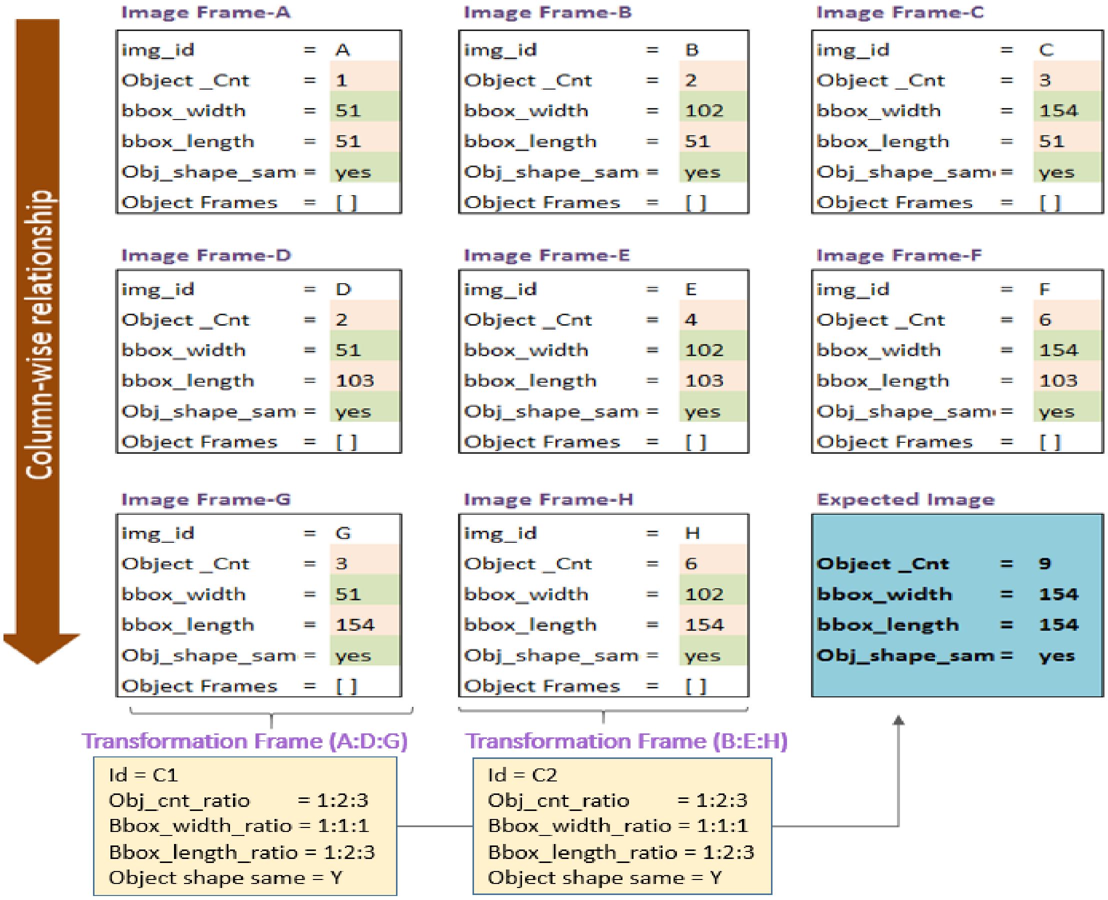


Image Frame-A

img_id	=	A
Object _Cnt	=	1
bbox_width	=	51
bbox_length	=	51
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-B

img_id	=	B
Object _Cnt	=	2
bbox_width	=	102
bbox_length	=	51
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-C

img_id	=	C
Object _Cnt	=	3
bbox_width	=	154
bbox_length	=	51
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-D

img_id	=	D
Object _Cnt	=	2
bbox_width	=	51
bbox_length	=	103
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-E

img_id	=	E
Object _Cnt	=	4
bbox_width	=	102
bbox_length	=	103
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-F

img_id	=	F
Object _Cnt	=	6
bbox_width	=	154
bbox_length	=	103
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-G

img_id	=	G
Object _Cnt	=	3
bbox_width	=	51
bbox_length	=	154
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Image Frame-H

img_id	=	H
Object _Cnt	=	6
bbox_width	=	102
bbox_length	=	154
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Expected Image

Object _Cnt	=	**9**
bbox_width	=	**154**
bbox_length	=	**154**
Obj_shape_sam	=	**yes**

Transformation Frame (A:D:G)

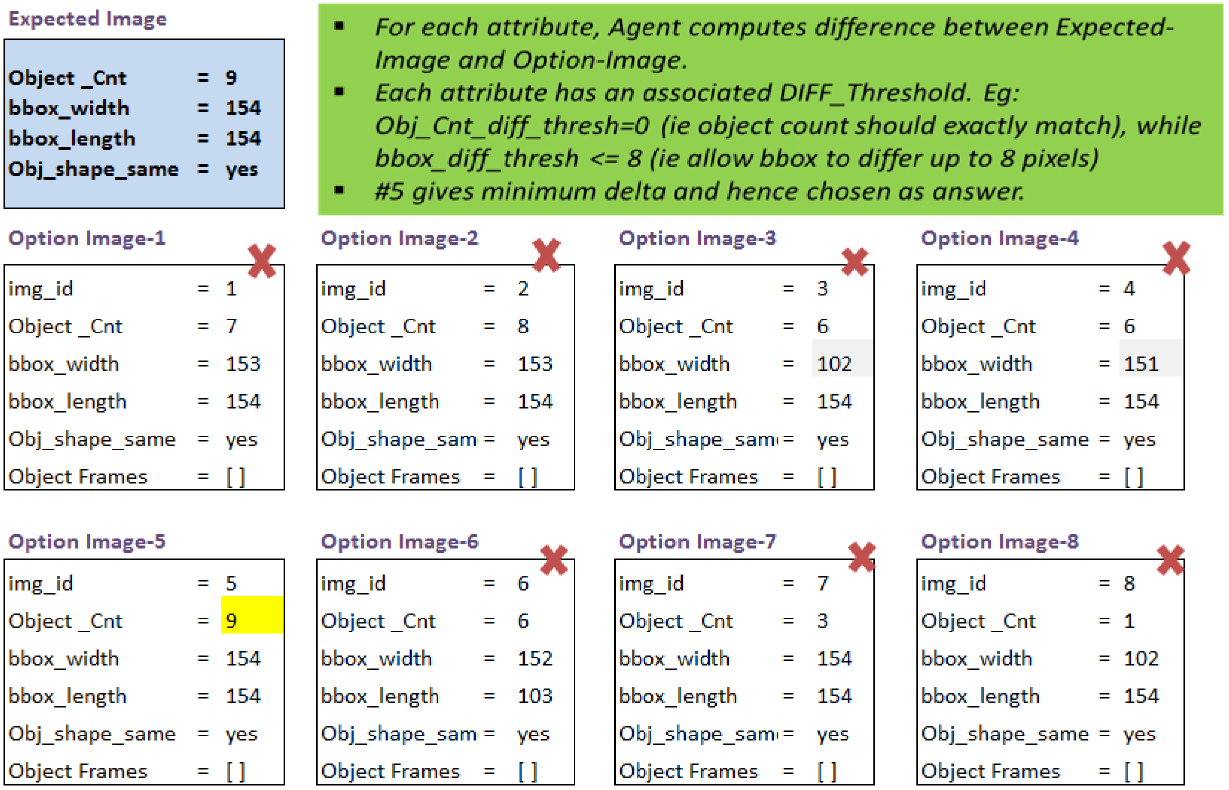
Id = C1
Obj_cnt_ratio = 1:2:3
Bbox_width_ratio = 1:1:1
Bbox_length_ratio = 1:2:3
Object shape same = Y

Transformation Frame (B:E:H)

Id = C2
Obj_cnt_ratio = 1:2:3
Bbox_width_ratio = 1:1:1
Bbox_length_ratio = 1:2:3
Object shape same = Y

Figure: 5c

Subtask 2.4: Find solution Image



Expected Image

Object _Cnt	=	**9**
bbox_width	=	**154**
bbox_length	=	**154**
Obj_shape_same	=	**yes**

- *For each attribute, Agent computes difference between Expected-Image and Option-Image.*
- *Each attribute has an associated DIFF_Threshold. Eg: Obj_Cnt_diff_thresh=0 (ie object count should exactly match), while bbox_diff_thresh <= 8 (ie allow bbox to differ up to 8 pixels)*
- *#5 gives minimum delta and hence chosen as answer.*

Option Image-1 ✗

img_id	=	1
Object _Cnt	=	7
bbox_width	=	153
bbox_length	=	154
Obj_shape_same	=	yes
Object Frames	=	[ ]

Option Image-2 ✗

img_id	=	2
Object _Cnt	=	8
bbox_width	=	153
bbox_length	=	154
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Option Image-3 ✗

img_id	=	3
Object _Cnt	=	6
bbox_width	=	102
bbox_length	=	154
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Option Image-4 ✗

img_id	=	4
Object _Cnt	=	6
bbox_width	=	151
bbox_length	=	154
Obj_shape_same	=	yes
Object Frames	=	[ ]

Option Image-5

img_id	=	5
Object _Cnt	=	9
bbox_width	=	154
bbox_length	=	154
Obj_shape_same	=	yes
Object Frames	=	[ ]

Option Image-6 ✗

img_id	=	6
Object _Cnt	=	6
bbox_width	=	152
bbox_length	=	103
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Option Image-7 ✗

img_id	=	7
Object _Cnt	=	3
bbox_width	=	154
bbox_length	=	154
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Option Image-8 ✗

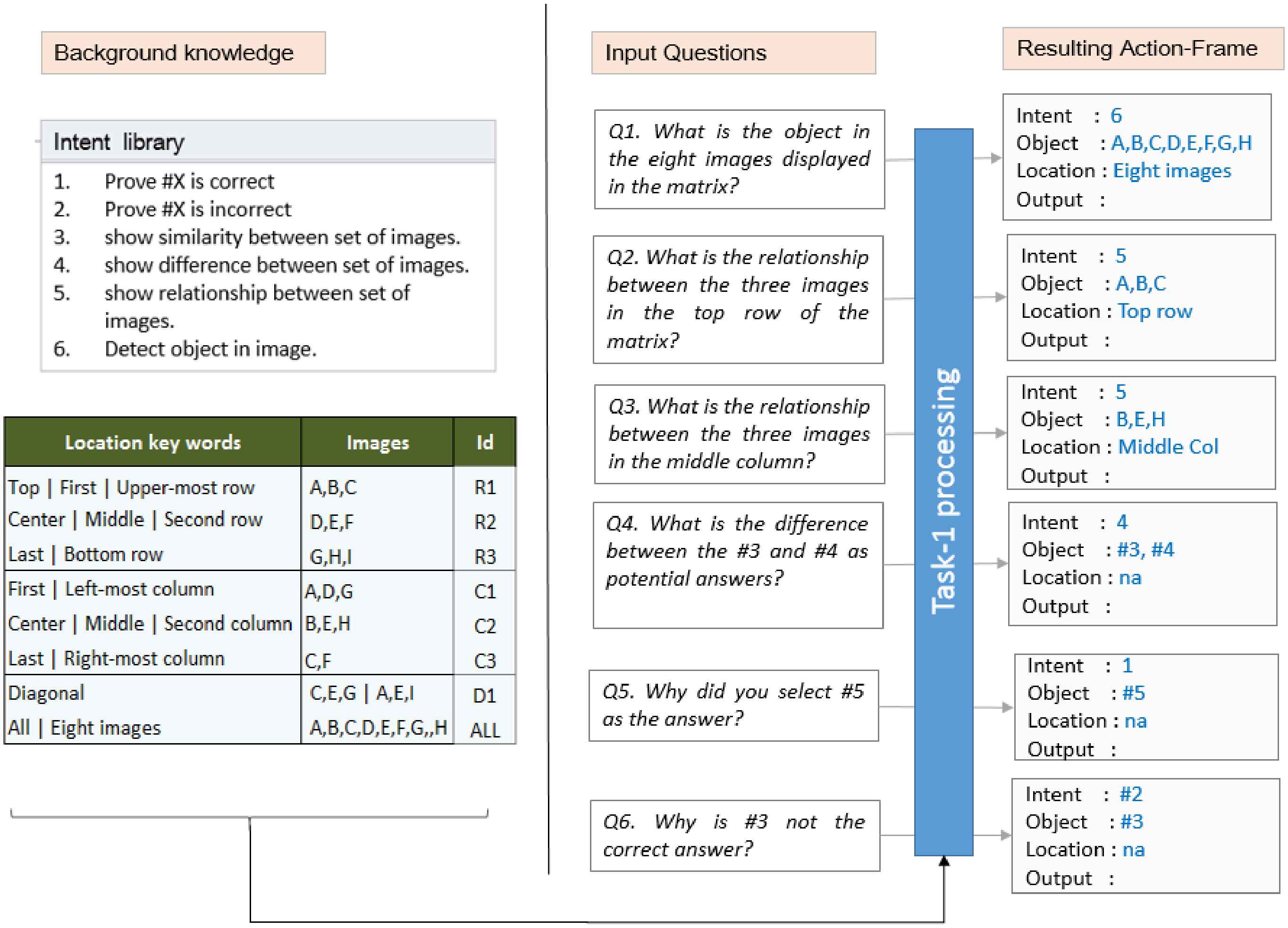
img_id	=	8
Object _Cnt	=	1
bbox_width	=	102
bbox_length	=	154
Obj_shape_same	=	yes
Object Frames	=	[ ]

Figure: 5d

Task 3: Generating Response (to user's query)

Armed with the knowledge from Task-1 and Task-2, Agent is now ready to trigger appropriate script that will create response for user.

- *Figure-6a below shows resulting Action-Frame (for each of the six questions) at the end of Task-1.*
- *Output of Task-2 is shown in Figure 5a-5d above.*



Background knowledge

Intent library

1. Prove #X is correct
2. Prove #X is incorrect
3. show similarity between set of images.
4. show difference between set of images.
5. show relationship between set of images.
6. Detect object in image.

Location key words	Images	Id
Top | First | Upper-most row	A,B,C	R1
Center | Middle | Second row	D,E,F	R2
Last | Bottom row	G,H,I	R3
First | Left-most column	A,D,G	C1
Center | Middle | Second column	B,E,H	C2
Last | Right-most column	C,F	C3
Diagonal	C,E,G | A,E,I	D1
All | Eight images	A,B,C,D,E,F,G,,H	ALL

Input Questions

Q1. What is the object in the eight images displayed in the matrix?

Q2. What is the relationship between the three images in the top row of the matrix?

Q3. What is the relationship between the three images in the middle column?

Q4. What is the difference between the #3 and #4 as potential answers?

Q5. Why did you select #5 as the answer?

Q6. Why is #3 not the correct answer?

Task-1 processing

Resulting Action-Frame

Intent : 6
Object : A,B,C,D,E,F,G,H
Location : Eight images
Output :

Intent : 5
Object : A,B,C
Location : Top row
Output :

Intent : 5
Object : B,E,H
Location : Middle Col
Output :

Intent : 4
Object : #3, #4
Location : na
Output :

Intent : 1
Object : #5
Location : na
Output :

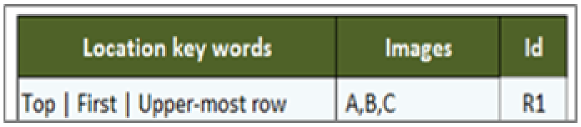
Intent : #2
Object : #3
Location : na
Output :

Figure: 6a

Q2-Q3

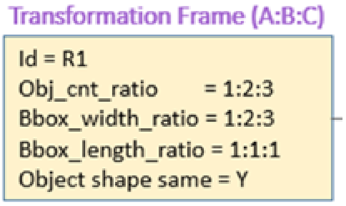
Agent is equipped with scripts that are indexed by Intent#. Let's say agent receives Q2. (Figure-6b)

1. Task-1 maps Q2 to Intent#5 ('show relationship') and 'Object' slot to A, B and C.
2. Task-2 populates all Image, Object and Transformation Frames (Fig 5a-5d).
3. Agent will retrieve script at index 5 from script-library and pass
   - *Q = Images mentioned in user's query, which are values stored in Object slot (A, B, C)*
   - *T = Images and Transformation Frames from Task-2*
4. From its background knowledge, Agent knows Top-Row maps to R1. It extracts Transformation-Frame for R1 and send this information to User Interaction Engine to display.



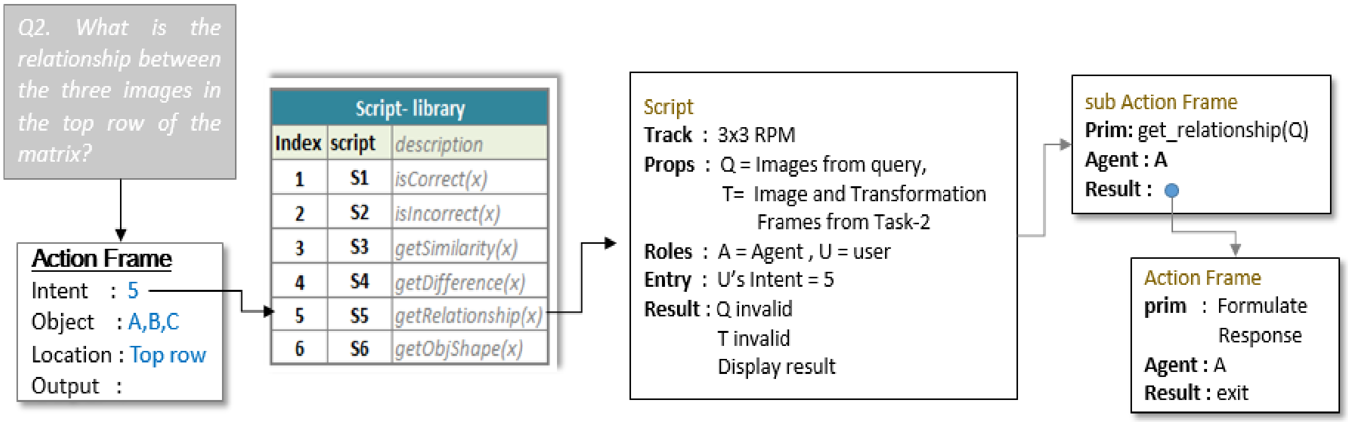
Location key words	Images	Id
Top | First | Upper-most row	A,B,C	R1

Transformation Frame (A:B:C)



Id = R1
Obj_cnt_ratio = 1:2:3
Bbox_width_ratio = 1:2:3
Bbox_length_ratio = 1:1:1
Object shape same = Y

5. Let's say if question was to about similarity or difference instead of relationship.
   - *Script will return only those attributes that didn't change (bounding box length and object shape in this example) if similarity was asked.*
   - *Script will return only those attributes that changed (Object count and bounding box width) if difference was asked.*
6. This approach allows agent to answer questions related to 'relationship', 'similarity' and 'difference' between any set of images.



Q2. What is the relationship between the three images in the top row of the matrix?

Action Frame
Intent : 5
Object : A,B,C
Location : Top row
Output :

Script- library

Index	script	*description*
1	S1	*isCorrect(x)*
2	S2	*isIncorrect(x)*
3	S3	*getSimilarity(x)*
4	S4	*getDifference(x)*
5	S5	*getRelationship(x)*
6	S6	*getObjShape(x)*

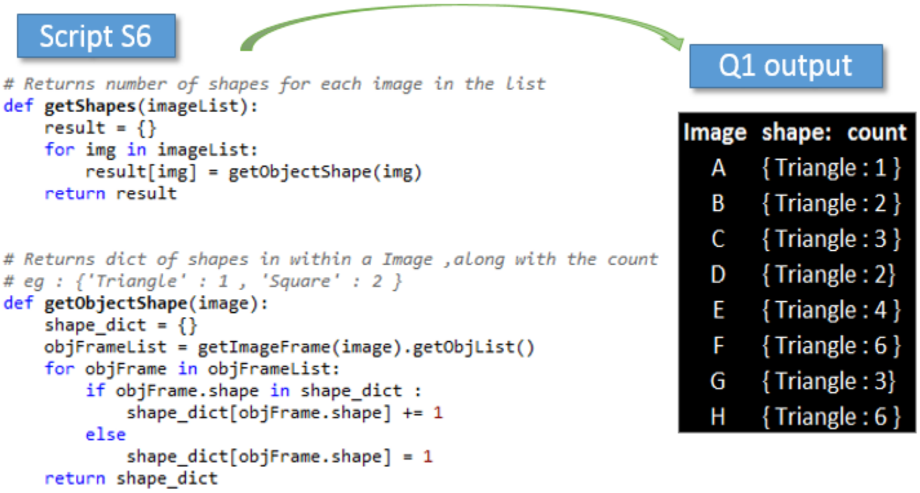
Script
Track : 3x3 RPM
Props : Q = Images from query,
T= Image and Transformation Frames from Task-2
Roles : A = Agent , U = user
Entry : U's Intent = 5
Result : Q invalid
T invalid
Display result

sub Action Frame
Prim: get_relationship(Q)
Agent : A
Result : ●

Action Frame
prim : Formulate Response
Agent : A
Result : exit

Figure: 6b

Q1 *what is the object in the eight images displayed in the matrix?*
This question gets mapped to Intent# 6, which invokes script S6, which iterates over objectFrame list for each image mentioned in user query and return shapes along with its count.



Script S6

```
# Returns number of shapes for each image in the list
def getShapes(imageList):
    result = {}
    for img in imageList:
        result[img] = getObjectShape(img)
    return result


# Returns dict of shapes in within a Image ,along with the count
# eg : {'Triangle' : 1 , 'Square' : 2 }
def getObjectShape(image):
    shape_dict = {}
    objFrameList = getImageFrame(image).getObjList()
    for objFrame in objFrameList:
        if objFrame.shape in shape_dict :
            shape_dict[objFrame.shape] += 1
        else
            shape_dict[objFrame.shape] = 1
    return shape_dict
```

Q1 output

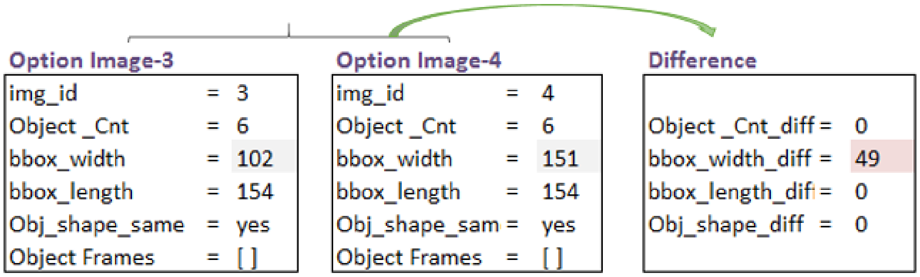
Image	shape: count
A	{ Triangle : 1 }
B	{ Triangle : 2 }
C	{ Triangle : 3 }
D	{ Triangle : 2}
E	{ Triangle : 4 }
F	{ Triangle : 6 }
G	{ Triangle : 3}
H	{ Triangle : 6 }

Figure: 6c

Q4, 5, 6

Agent generates expectation for solution image (Figure 5b and 5c). As illustrated in Fig-5d, Agent determines which option-images did/didn't satisfy the constraint of expected transformation and hence can answer why a certain option-image is correct or incorrect answer.

Q4. What is the difference between the #3 and #4 as potential answers?
Agent checks the attributes that differ in image #3 and #4, which shows bbox_width of #4 is 33% (49/151) larger than that of #3 and returns this as answer.



Option Image-3

img_id	=	3
Object _Cnt	=	6
bbox_width	=	102
bbox_length	=	154
Obj_shape_same	=	yes
Object Frames	=	[ ]

Option Image-4

img_id	=	4
Object _Cnt	=	6
bbox_width	=	151
bbox_length	=	154
Obj_shape_sam	=	yes
Object Frames	=	[ ]

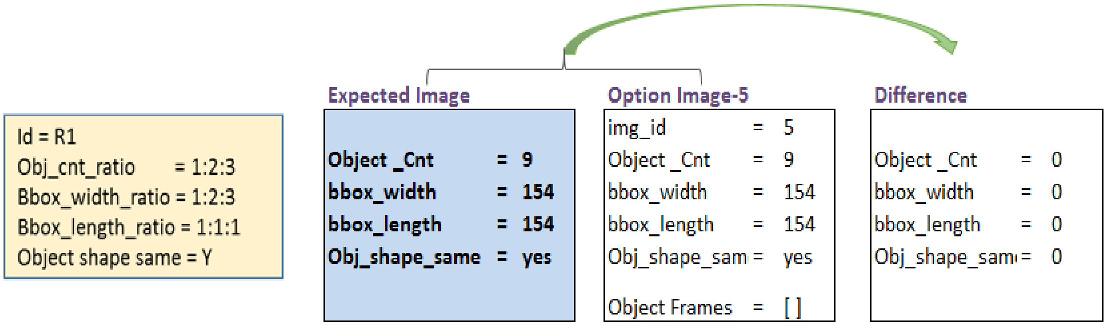
Difference

Object _Cnt_diff	=	0
bbox_width_diff	=	49
bbox_length_dif	=	0
Obj_shape_diff	=	0

Q.5 why did you select #5 as the answer?

Agents knows expected Image attributes, based on transformation observed in Row-1, 2 and Col-1, 2 from Task-2. #5 is the only image that matches up with expected image.

- Agent's response: *"#5 shows least amount of difference from Expected image, hence chosen as the answer*."
- Agent also displays attributes of Transformation-Frame, Expected-Image frame and difference (shown below)



Id = R1
Obj_cnt_ratio = 1:2:3
Bbox_width_ratio = 1:2:3
Bbox_length_ratio = 1:1:1
Object shape same = Y

Expected Image

Object _Cnt	**=**	**9**
bbox_width	**=**	**154**
bbox_length	**=**	**154**
Obj_shape_same	**=**	**yes**

Option Image-5

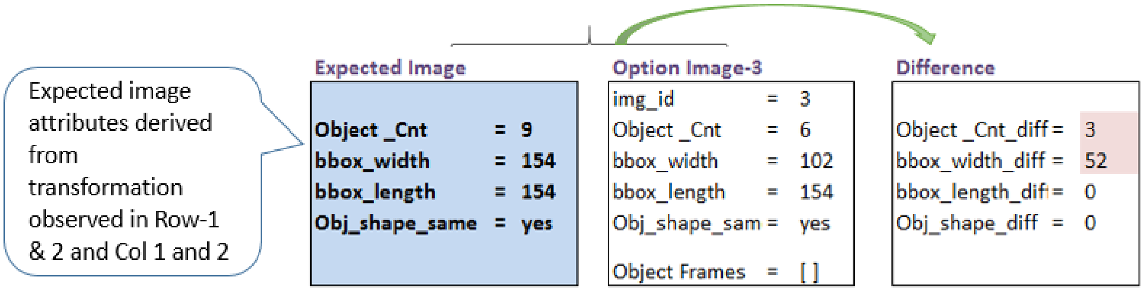
img_id	=	5
Object _Cnt	=	9
bbox_width	=	154
bbox_length	=	154
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Difference

Object _Cnt	=	0
bbox_width	=	0
bbox_length	=	0
Obj_shape_sam	=	0

Q.6 Why is #3 not the correct answer?

Similar approach as above. Agent returns the differences between #3 and Expected image as reason for #3 not qualifying as the correct answer.



Expected image attributes derived from transformation observed in Row-1 & 2 and Col 1 and 2

Expected Image

Object _Cnt	**=**	**9**
bbox_width	**=**	**154**
bbox_length	**=**	**154**
Obj_shape_same	**=**	**yes**

Option Image-3

img_id	=	3
Object _Cnt	=	6
bbox_width	=	102
bbox_length	=	154
Obj_shape_sam	=	yes
Object Frames	=	[ ]

Difference

Object _Cnt_diff	=	3
bbox_width_diff	=	52
bbox_length_dif	=	0
Obj_shape_diff	=	0

[Note: I have shown response for above questions in graphical format for ease of understanding. Agent outputs response (frame attributes/ differences etc.) in text format, similar to Bonnie's output]

Handling of unknown/error cases

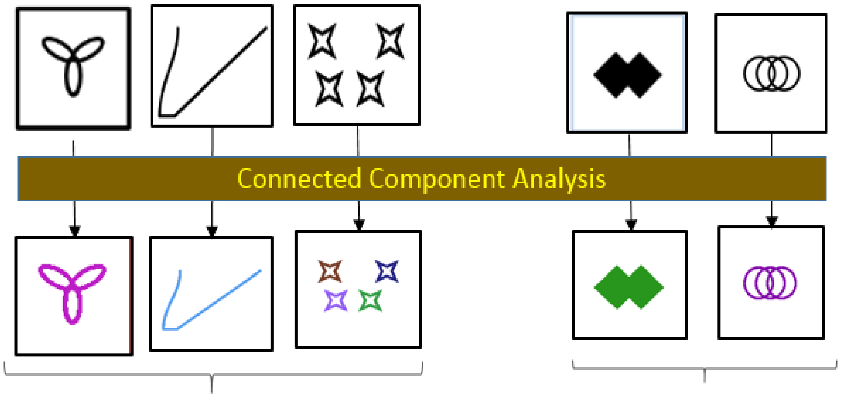
Agent will generate a canned response

- If Agent is not able to map user's query to an Intent (from its preconfigured library).
  - *Eg: user: "Why #3 is so blur?"*
  - *Agent: "sorry I can't answer that question."*
- If image# extracted from user query is not within expected range.
  - Eg: *user: "Why #14 is incorrect?"*
  - *Agent: "Invalid Image# in query. Expected Images are A -H and 1-8."*

Limitation and Enhancement

1. **Web Based Interface:** Though this assignment asked to us to use text-based interface for user interaction, I strongly believe allowing visual response will help the purpose greatly. My agent's shape detection is its weakest point (Figure-7)
   - *It makes mistake when objects are overlapping.*

- *It can't classify odd-shaped objects or the ones for which is there no easy way to compute area formula like star, heart or a flower.*



Agent can extract individual objects but is not able to describe it to user.

Agent fails to extract individual object as objects are overlapping. Eg : It marks 3 overlapping circles as one object.

Figure: 7

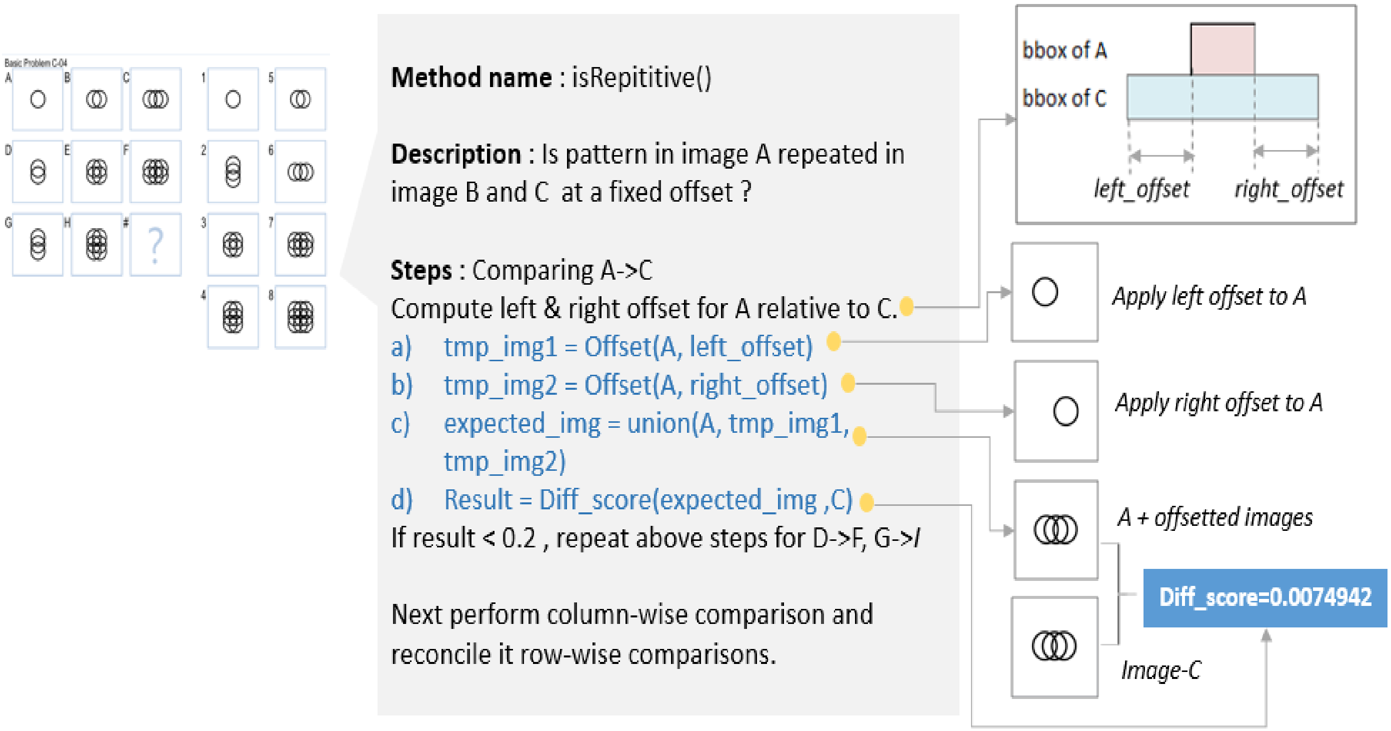
I propose to enhance user interface to be web based so that Agent has option to provide visual response for cases where

- *Agent's processing is shape agnostic.*
- *Agent is able to extract the object but can't put it in words (unless agents trained to describe complex/odd images , which requires time and effort)*

Shape agnostic processing - In cases where agent is not able to derive an answer by shape extraction, it will fall back to shape agnostic processing. I will explain this with an example that is similar to the problem given in Final-Exam document.

In example below, Agent's shape extraction logic fails as objects are overlapping. So it tries a generic shape-agnostic transformation described in figure below. This method is generic enough to handle Fig-5a problem as well other problems like Basic C-4, C-6, C-10, and Challenge C-2.

While agent can't describe the object, it can very well explain its logic at each step with help of images as shown on the right.



Method name : isRepititive()

Description : Is pattern in image A repeated in image B and C at a fixed offset ?

Steps : Comparing A->C

Compute left & right offset for A relative to C.

a) tmp_img1 = Offset(A, left_offset)

b) tmp_img2 = Offset(A, right_offset)

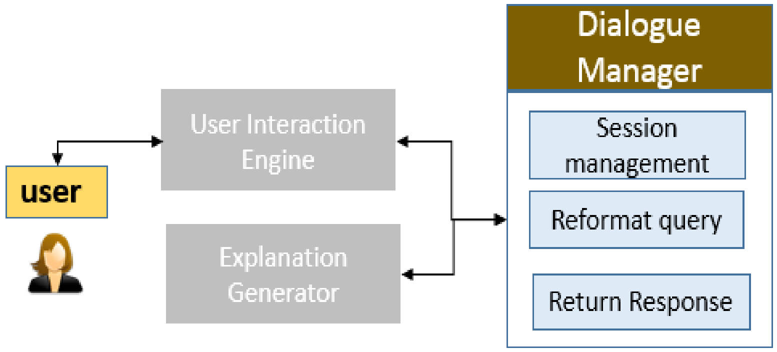
c) expected_img = union(A, tmp_img1, tmp_img2)

d) Result = Diff_score(expected_img ,C)

If result < 0.2 , repeat above steps for D->F, G->I

Next perform column-wise comparison and reconcile it row-wise comparisons.

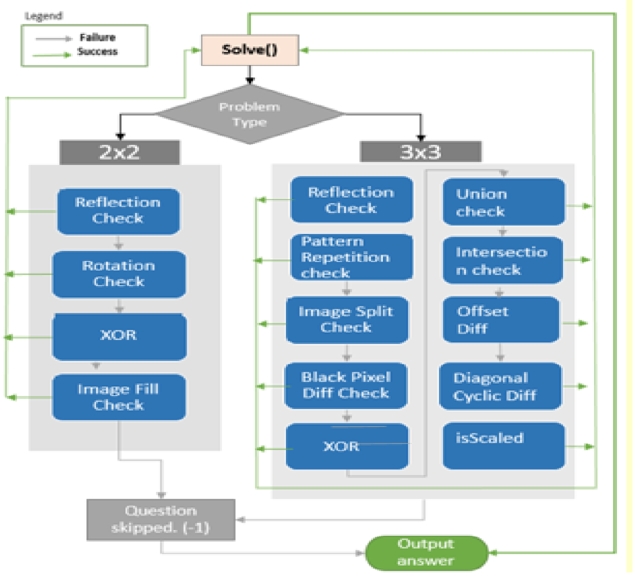
2. **Discourse Coherence**: Current design is built for Question-Answer type interaction. Agent can't carry out conversation. This can be achieved by introducing a new functional module called 'Dialogue Manager' which will interact with User-Interaction module to Explanation-Generator to fill this gap.



- *Creation & deletion of dialogue session for each unique user.*
- *Keeps track of requests/responses exchanged in the session to learn context of the conversation*
- *Reformat query if needed – resolves referent.*
- *Avoid circular exchanges (by lowering the weight of a response with each use.*
- *Forwards this query to Explanation Generator*
- *Formulate Response to user's query*

3. Current design can be augmented to allow Agent to answer question related to its Transformation knowledge base. Eg:
   - User can ask "*What all transformation type do you know*"?
     Agent can display description of all transformation modules in its library.

   - User may ask *"Why Transformation# X is not a valid transformation for this problem?"* or *"What all transformations did you try for this problem?"*
     As illustrated in Figure-2b, Agent knows sequence of transformation it is configured with (blue boxes in snap shot below). Hence it can display Transformation-Frame generated when a given transformation was tried. It can also explain why a particular Transformation was not valid, if it didn't yield an answer within desired confidence threshold.



Conclusion

Dystopian narratives by Hollywood movies and some business leaders have made many people not to trust AI systems. Simplicity of design presented in this paper makes up for its ease of implementation and is a small step towards gaining that trust.

References

1. Goel, A. & Joyner, D. (2016). KBAI eBook: Knowledge Based Artificial Intelligence. *Retrieved from https://files.t-*square.gatech.edu/access/content/group/gtc-c01a-0bd6-5b67-9c88176eb86d0533/KBAI%20ebook/kbai_ebook.pdf

2. Chauhan, Arti (2016) , CS 7637 Project-3 Reflection - unpublished paper

3. Stanford Parser and universal dependencies
   - http://nlp.stanford.edu/software/stanford-dependencies.shtml
   - http://nlp.stanford.edu/software/lex-parser.shtml