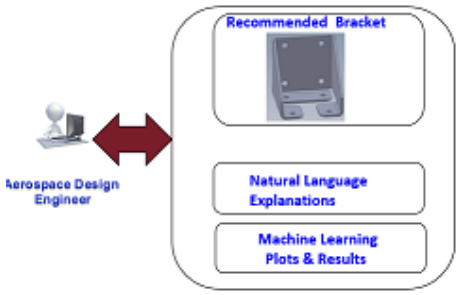
Explainable Artificial Intelligence for Raven's Progressive Matrices

Introduction:

Explainable Artificial intelligence systems are a cornerstone design principle of knowledge based recommendation systems aimed at advanced users. For example, an aerospace engineer routinely leverages knowledge based recommendation systems to select appropriate bracket (a standard part) while assembling airframe structures. An explainable expert system recommending the bracket would enable reactive natural language conversations, graphs and figures aimed at backing up an agent's recommendation of the bracket. The agent's explanation enables an engineer to weigh on decisions where the stakes may be as high as affecting passenger safety. In such industrial scenarios, the clarity of the agent's explanations are as important as the correctness of the recommendation itself.

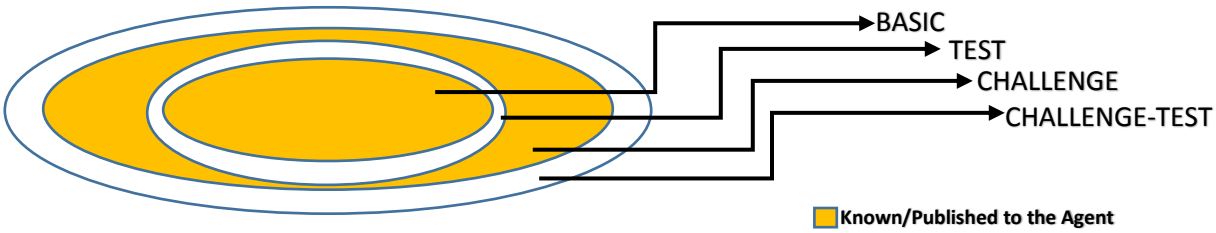
Analogy from aerospace industry



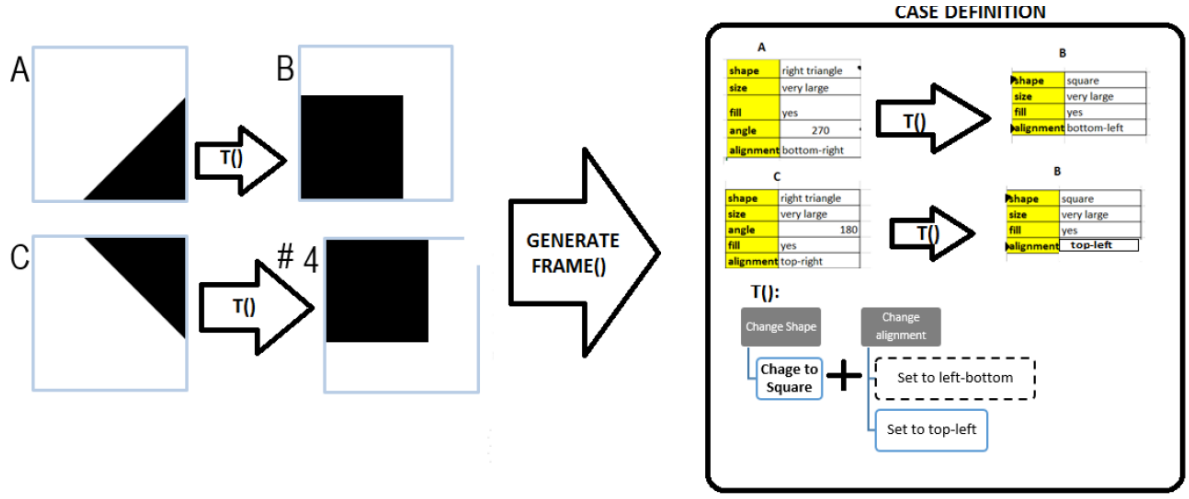
Raven's Progressie matrices

We are about to design an AI agent to solve Raven's Progressive matrices problem.Let us assume that the seriousness illustrated in the real world example somehow applies to agent's ability to explain the RPM solution.The agent is expected to correctly choose the answer from the provided set of six choices. We are provided with a set of problems (Basic/Challenge). We are expected to design an agent that can solve unseen "Test", "seen" problems and being capable of interactions with a user to answer the following questions.

`



Solving the RPM: We will use a case based reasoning and adaptation approach to solve the RPM. Subsequent sections will detail the choice of a case based aproach benefiting the development of an explanative interface



Let us understand the structure of a simple solved problem before proceeding to problem solving methods. A **case** is a frame based semantic network either generated or available through the verbal representation for any given basic problem. The case definition primarily comprises two components:

Frames For Problem Figures:
A part of the **case** characterizes/encodes the states of all RPM figures using shape, size, fill status, alignment etc. and also the transformation functions **T ()** associated with them.

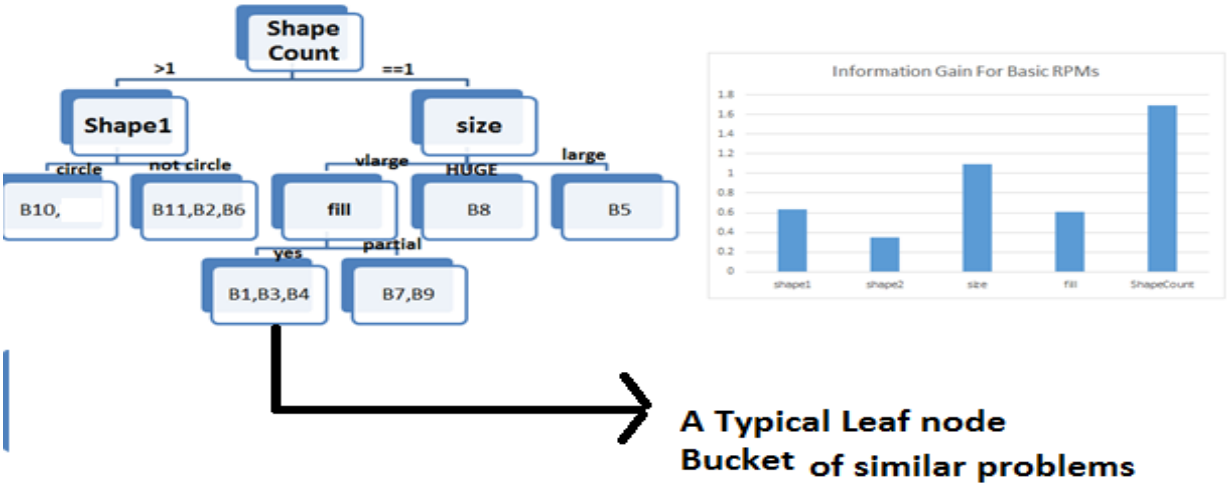
Transformation representation:
Since we aim to store the case to solve a "**similar**" problems and not just the exact problem, the case's **T()** definition may contain structures that explicitly bring about the specificity of the operation applied to the figure. The **T ()** representation in the above diagram also contains a dotted "**Set-to left-bottom**" whic***h would have been the transformation deduced*** had the bottom left figure (**#4**) not been in the given choices.

Decision Tree Learninig

We leverage a specialized discriminant tree approach somewhat different from to perform case indexing. The primary intuition to leverage a stochastic technique is to improve the generality of the RPM solver. We define similarity between any two **cases** as the graph distance between transformation functions of those cases. We aim to build a probabilistic model for the decision tree that would hash any given RPM to a leaf node which represents the sub-category of RPMs that are most **similar** to the input **case**. Hence, the decision tree computes problem attributes that are detrimental to effectively splitting the **case** space along **similarity** dimension. A decision tree algorithm probes the various dimensions of the **"Case"** using **entropy** and uses information gain to generate a decision tree as shown below. The decision tree categorizes **Cases** into linear buckets stored at the leaf of the tree. Hence a new case to be solved would be hashed to the closest bucket that contains the most –similar sub category of problems within the RPM distribution.

Reasoning on Cased Based Storage

The decision tree technique places the most significant discriminators at the top of the tree which increases the likelihood of O (log n) retrieval speed of a case from storage. The impact of decision tree on subsequent case adaptation is discussed below. Once the input case is hashed to its bucket of similar problems, we move on to actually retrieving the most **similar** case from the bucket. We compute the ***confidence (see definition below)*** of similarity across images that are likely similar and retrieve the **case** with the most confidence value. Such an optimization is likely to minimize the search space for any randomizations in case adaptation phase. Let's assume that B12 shown above is the case that must be solved given the decision tree above. B12 hashes to the same bucket as B10 by traversing the tree. The idea of the decision tree is to classify transformation similarities into buckets and improve average time complexity on case retrieval and adaptation. Since the bucket has only one element, we skip the confidence computing phase and B10 is returned as the output for case retrieval phase.



$$confidence = 100 - \frac{different\ pixels * 100}{total\ pixels}$$

Case Adaptation by Recursion

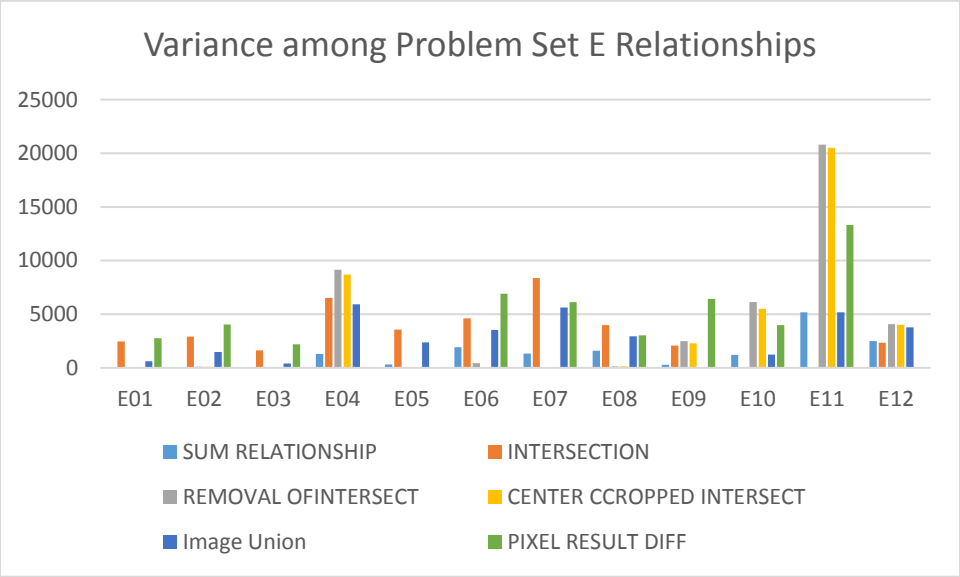
Once a case is retrieved from the tree, the **T()** function inside that case gets optimized. We apply discrete optimization algorithms like hill climbing to optimize the search for the adapted transformation. In the above example, once B10 is retrieved from memory, we apply hill climbing technique to search for that transformation which will solve the RPM (A:B for B12). To know if the computed transformation solves the problem. The hill climbing methods involve random restarts to avoid getting stuck at local hen the above case for B-10 is retrieved as explained before, the T() function shown above is adapted to T1() which is necessary to solve the input RPM. Since we are using optimization techniques, we have to evaluate case performance not after the adaptation process is complete. We apply any intermediate T1() to Figure A in the input problem (B-12) measure the confidence metric for any intermediate transformed figure in hill climbing algorithm to detect if the transformations are

Design of Explanation Generators broadly fall under two categories:

1) Template Based Explanation Generators
2) Reactive Explanation Generators

Template Based Explanation Generator

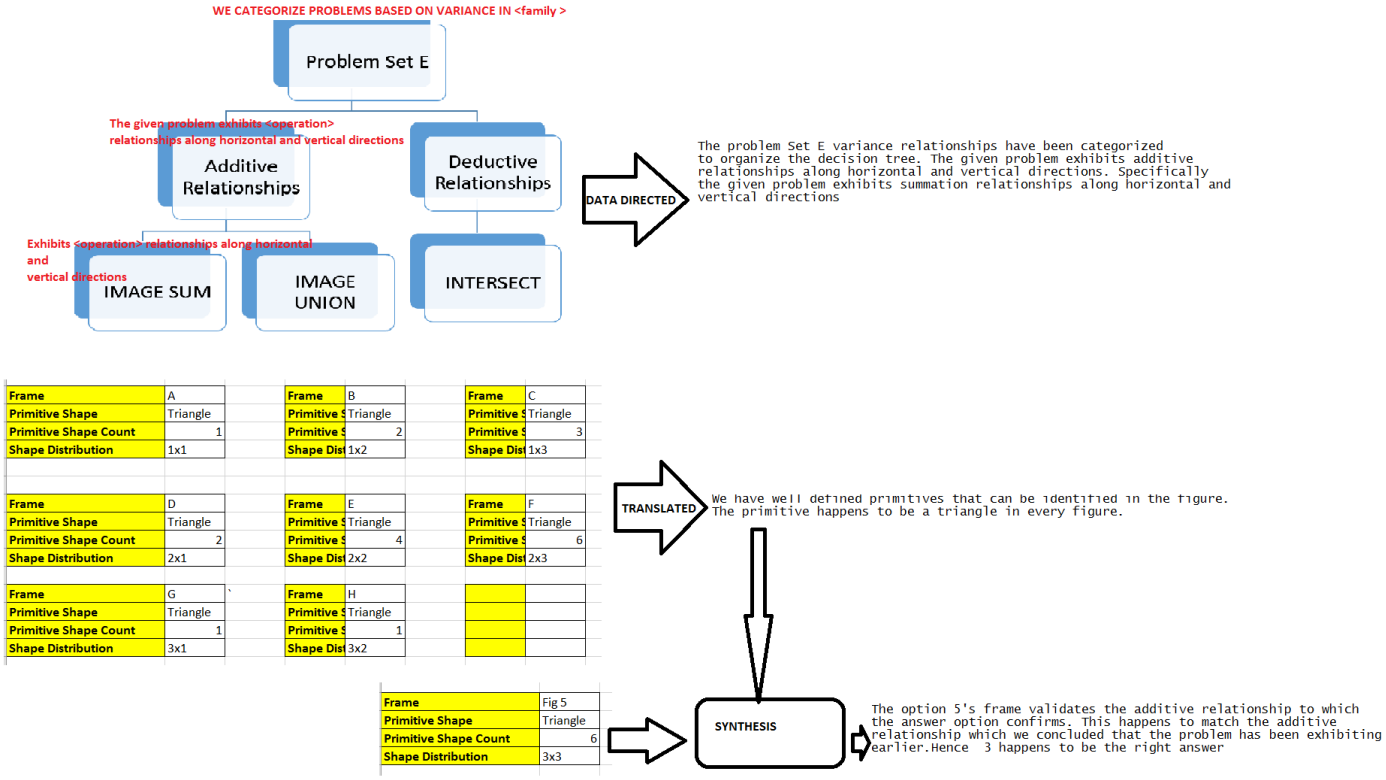
The design of an explanation generator could start as simple as the ones that fill out predefined template inorder to provide explanations. The parameterizable templates serve as components of domain knowledge explanation that can by itself illustrate a desilearning process (decision tree generation) to output explanations. This helps illustrate design principles in generic terms as applicable to a family of problems. The templates themselves rely upon reasoning the However during generation such parameterized templates are subtituted with specific problem data which enable understanding how such principles are applied in the context of the given problem for which reasoning is done. Below, we apply a data directed translation method to generate static template based explanations for the given problem.



In the below illustration, we have built a case based discriminant tree for the problem Set E and we use the variance chart shown above to construct a case based discrimination tree that solves the family of Set E problems. Here are some of the techniques we use to generate these explanations

Data Driven Translation : In the below diagram of a case based decision tree, by affixing generic natural language semantics to each tree node(see red text below) , ***the case indexing process will incidentally generate specific meta reasoning on the learning performed by the agent for the specific problem at hand***

Natual Language Synthesis: We can establish causality relationships between the horizontal/vertical relationships and a specific answer choice being evaluated to be correct or wrong.

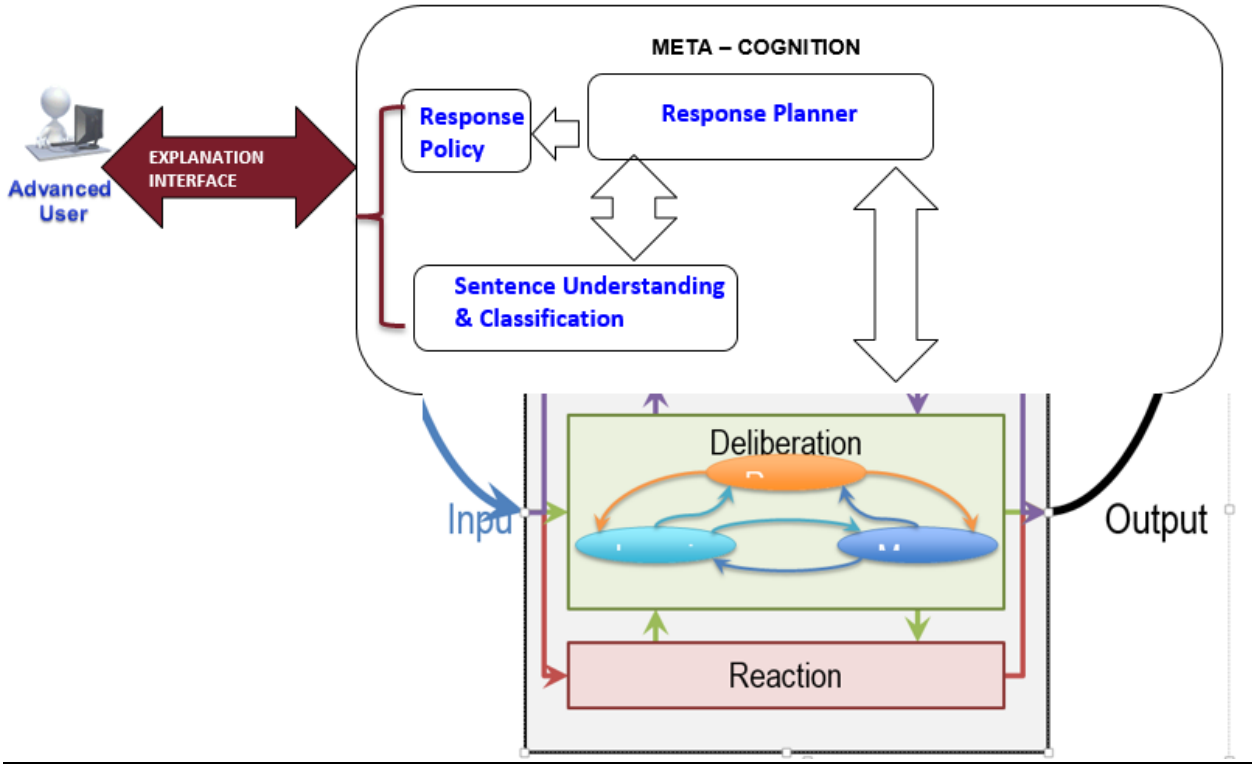


Meta cognition of the decision tree learning helped understand that aspect of the given problem along which the problem exhibited maximum variance (additive pixel relationships) among most other problems of the family. Meta cognition of the stored frame representations for each problem figure (Memory) helped understand shape primitives present in the figure. A deterministic analyses of the above meta cognition output helped establish causality to justify correct/wrong answer choices. ***Using structured formalized representations of problem solving strategy and figures in the problem data, we can generate meaningful explanations on the learning, reasoning and memory of the agent by augmenting the frames with semantic templates.***

Drawbacks of Template Based Explanations

The main drawback of template based explanation is that it does not emulate an interrogative human understanding model. Either toolittle information or too much information as per the predetermined process is put forth to the user and may end up confusing or leaing an incomplete impression of the solution to the user. This does not targetedly address the knowledge gaps encountered by every user evaluating the answer choice and problem relationships. The differential requirement for explanations may also arise when multiple users have differential background knowledge require to have specific concerns addressed about the presneted solution.Hence we discuss a reactive approach to generating the explanation.

Reactive Meta Cognitive Explanation



An agent that build reactive explanations respond to targeted queries to the user optimally based on past answer history and the agent's perception about the user's background knowledge. However, the agent goes through a planning phase when the problem to be solved is selected by the user. The planning phase outputs a "policy" in which the agent computes the optimal explanations to be provided given a specific user question and past questions. Upon every user question, the policy is consulted and the optimal explanation for a given context is generated by the agent.

Planning by Reinforcement Learning States, Actions and Rewards

We can use reinforcement learning approaches to implement planning performed by the agent. Below we outline an approach to modeling the reactive explanation generation as a Markov Decision problem. The problem can be solved by several well known approaches.

States: A state represents the level of knowledge imparted by the agent to the user. The extent of knowledge could vary by breadth and could vary by depth leading to infinite possible continous state space that must be used. However, we can discretize the states based on reasonable assumptions about organizing the state space. Such assumptions could lead to a meta cognitive layer whose generality is bounded within the ability to explain only those problems similar to the input problems (like Basic SetB,C, D & E)

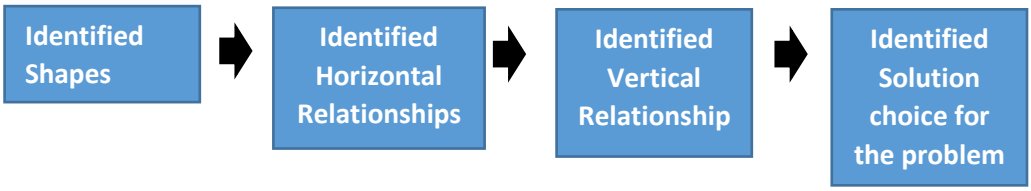
States covering Knowledge Breadth

User: What are the shapes that I see in the above figure ?
Agent: Those shapes are traingles in every figure

User: How do you explain the horizontal relationships?
Agent: The Horizontal relationships have incremental shapes?

User: How do you explain the vertical relationships?
Agent The vertical relationships have incremental shapes too.

User: So, what is the solution to this problem ?
Agent: The solution to this problem is (5) since it vindicates our assumption about horizontal and vertical relationships



These are the states that span the width of the problem solving process. So far, this is similar to the template generation process explained in the previous section. Finally the agent may use causality in the identified relationships to justify the correct answer choice. It is important to note that states above are modeled with strict predecessor states. An agent which attains a state of successfully having explained the asnwer choice certainly has also explained the relationships prior to which explained the identity of primitive shapes. This is called the Markov property and is reflective of real world conversational understanding.

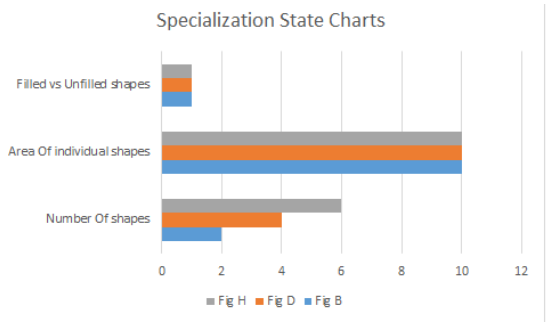
States covering Knowledge Depth

However, a user may or may not necessarily stand convinced about any given explanation by the agent. Different users will have multiple thresholds of necessary detail for convincing themselves of having understood a concept. Hence, once an agent attains a state by uttering the explanation a user query might instigate further depth and detail about the same specific phase which the agent has just finished explaining. We need to model such specialization states, so they can be generated during the planning phase.Every **Breadth covering state will have one or more depth states that indicate the level of detail with which the knowledge has been presented to the user.Since the agent is reactive in nature, such information is not presented prior to the user explicitly requesting the detail**

User : What relationships do the middle row figures indicate ?
Agent: The middle row figure indicates an increasing additive relationship in area
User: Can you please explain ?

The above scenario warrants the agent moving to a specializarion of the state which identified vertical relationships

Agent: Ok. Take a look at the below chart that explains the variance in relationship parameters among vertical column figures



There are two important takeaways here:

1) Explanations have no reason to be bounded within natural language. In-depth specializzed explanations may very well leverage graphical artifacts in modeling their utterances. Such implementations can use grammar based graphics that are available in implementations (ggplot) and can enable the user to build a graph layer by layer depending on the extent of depth required for explaining a concept.
2) Sentence understanding module comprise classifiers are capable of discriminating between user requests about wanting to understanf an aspect of the problem solving process from requesting additional detail for rationalizing an already explained concept.

States covering Knowledge Precedence

The user might start a conversation with the question about an answer choice. This could be due to the user posessing background knowledge or the user not knowing about the existence of prerequisite states. Hence we need more states in our state space that quantify the variance of prerequisite knowledge that the user posssess before understanding the agent's explanation

Identifying answer choice to the problem		
Horizontal relationships	Vertical relationships	Primitive Identity
1	1	0
1	0	1
0	0	0

We can use a bit vector to generate these states. Each prerequisite state is indicated by a vector whose value is one if the knowledge about that prerequisite is already available with the user as shown above

Actions:

The actions that are taken by the agent are responsible for progression of the user's knowledge from one state to another. The actions can be used to incidentally generate the explanations as shown in the template based architecture. However, in a reactive context, the actions are broadly classified as follows:

Breadth Progression Actions: These actions enable the agent to generate an explanation that signifies a breadth wise progression in fully understanding the problem

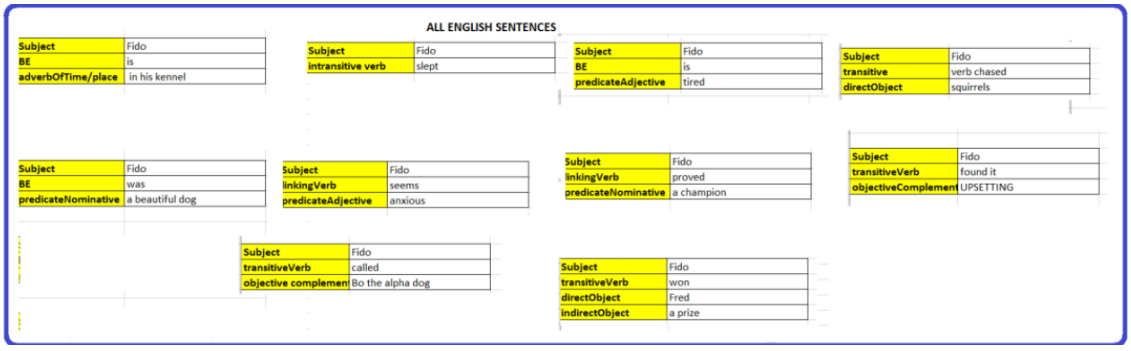
Depth Progression Actions: These actions enable an eagent to progress depthwise about the same state that was just attained after utterance of an explanation

Precedence Redirect Actions: These are actions taken by the agent due to a user being in a state without prerequisite background knowledge. This may require redirect appropriately for the user to model the knowledge of the precedence steps since we have them documented in the bit vector if that state.

Rewards: Reward functions emulate immediate feedback from the environment to indicate how the action has impacted the user's knowledge or understanding. For example, lets take the case of a user posing an answer explanation question without the background knowledge of horizontal/vertical relationships. A redirect action taken to move to its prerequisite states will carry a high positive reward. An action taken to explore further depth about the asnwer itself will carry a nnegative reward.

Sentence Understanding module

The sentence understanding phase plays a key role in observability of a state by the agent. The agent's history of questions answered and the correct classsification of an upcoming user question combinedly help decide on the state of user's understanding. We can use several natural language design mechanisms to design the sentence understanding module. The module will convert an input sentence from the user to a structured representation in any one of the tens sentence forms to be able to classify its significance



References:

http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=87686
http://www.darpa.mil/program/explainable-artificial-intelligence
https://pdfs.semanticscholar.org/44ff/6701d0ca7685b4d51f153ef8888fe2c2de2d.pdf