Zachary Waters

Ashok Goel

CS 4635 Knowledge-Based AI

December 3, 2019

Final

Part A:

A knowledge representation acts a surrogate for some real-world problem/knowledge that allows a person to act on it through internal mental calculations instead of external actions acting. There are several factors that determine how you select the best representation for the knowledge/problem. A primary characteristic of a good a representation is that it hides the irrelevant information while at the other end, making the pertinent information explicitly clear, such as any natural constraints. Secondarily, it may be important that your chosen representation be efficiently computable, so that an artificial agent can conduct its own calculations. There are many other tertiary characteristics you may want to try and achieve with your representation, such as having the ability to easily incorporate new information into its own structure in an efficient manner.

Part B:

- **Generate & Test and Diagnosis:**

Generate & Test is the process whereby using a generator and a tester, you create many potential solutions to a problem. Diagnosis is the process where you look at a malfunctioning entity and through examining its various "symptoms" you create a plausible hypothesis that explains the cause of the malfunction. You test your hypothesis by comparing what the known symptoms with the expected symptoms of the hypothesis though the principles of coverage and parsimony. This means that diagnosis can be seen as a type of generate and test, where you the hypothesis you generate are the solutions to the problem of "what caused this malfunction".

- **Planning and Search:**

Planning is the process of determining a sequence of steps that are needed to be taken in order for the agent to reach some goal state. In planning the steps are often have structured representations that require the agent to have some heuristic system in order to account for them. Search however is a much broader technique where you examine a set of states are looking for some particular subset of them. This is usually used to navigate from a start state to some goal state. Planning can be thought of as a type of search, where you search through the space of possible actions in order to find some subset of action sequences that meet the criteria.

- **Analogical Reasoning and Case-Based Reasoning:**

Analogical reasoning is the process where you use previously seen groups/families of problems in order to solve new problems. It does this by using the knowledge of known relationships as its base of transfer, allowing analogical reasoning to be used on problems across vastly different domains. Case-based reasoning similarly uses previously recorded problems in order to solve new problems. However, what differentiates case-based reasoning is that the transferable domain between the target and source questions is limited by not only the relationships, but the objects as well. This means that case-based reasoning is much more limited in its scope of application and often requires having many more stored examples order to anticipate and solve new problems. This means that we can see case-based reasoning as a type of analogical reasoning that is more limited in its scope of comparison.

- **Semantic Networks and Frames:**

Semantic networks use a graph structure in order to represent the relationships between various concepts. Each vertex of the semantic network represents a concept, and the edges that connect these vertices establishes an explicit relationship between the connected concepts. Frames can best be compared to a table of information. Each table consists of "rows" called slots. These slots are substructures of the frame that contain the various attributes that comprise the totality of the concept they are trying to represent. Frames also allow for an easy implementation of inheritance through their ability to have a "default" slot-filler value for their various slots. We can use semantic networks and frames together to create a much more robust and detailed representation network, where the nodes are now replaced by frames. This new system allows us to imbue the various elements of the semantic network with their own sets of attributes and can now also incorporate frame's ability of inheritance in order to create hierarchical structures as well.

- **Metareasoning and Configuration:**

Metareasoning is a reasoning strategy that examines the internal processes of the agent itself. This can be used for a variety of effects that include completing gaps in its own knowledge, correcting mistakes that may have occurred during the learning or reasoning processes, and determining what strategies to use for learning and reasoning. Configuration is the process of determining what variables to use in order to satisfy the constraints of a multi-variable problem. The strategy selection aspect of meta reasoning can be thought of as a higher-level version of a configuration process. This is because the metacognition system will often have to compare many different aspects of the various strategies it may use, such as their input-informational requirements, their computational efficiency, or their quality of solutions; all in order to determine what sequence of processes will best lead to the goal state.

- **Metareasoning and Learning by Correcting Mistakes:**

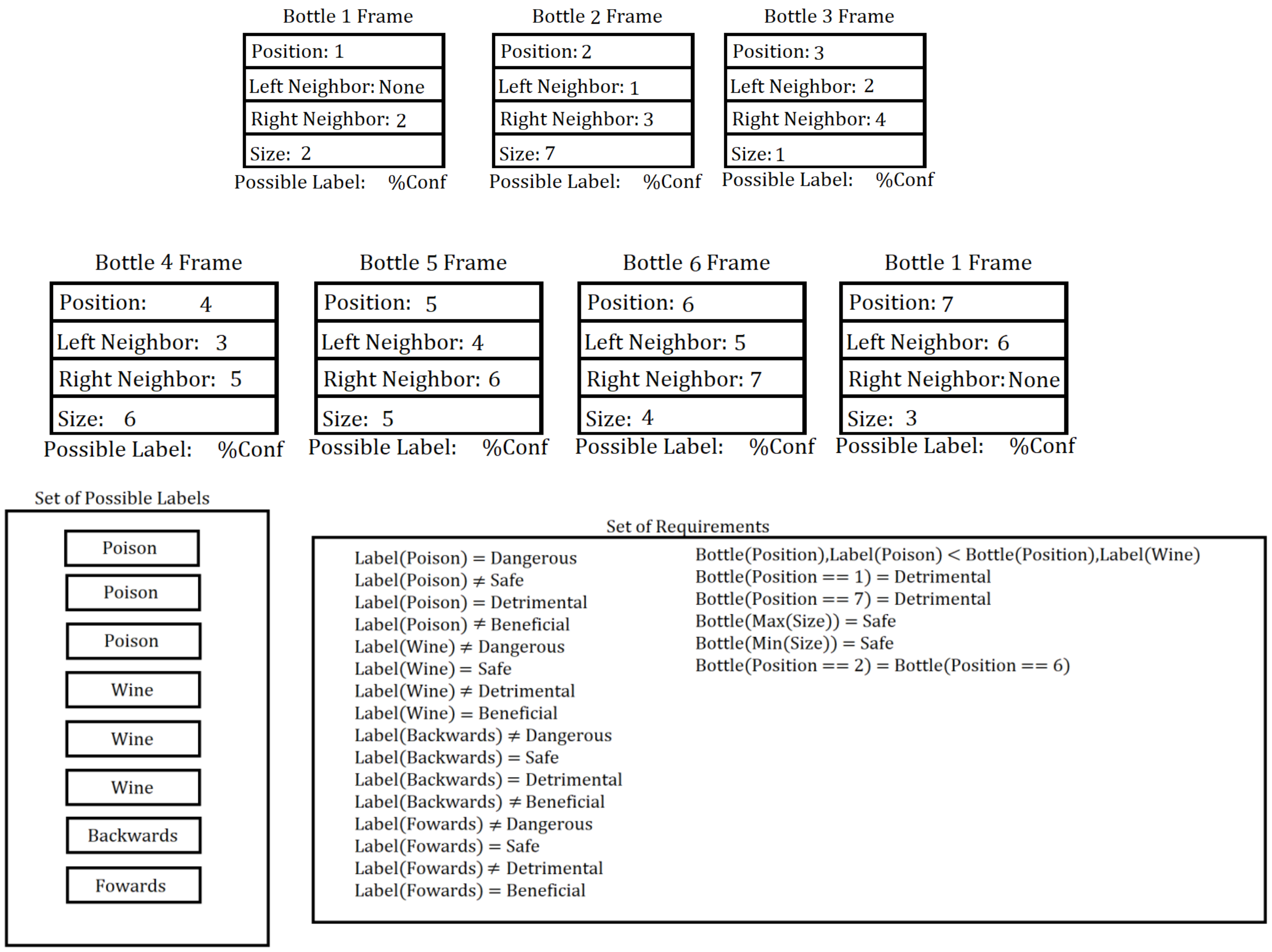
Metareasoning is a reasoning strategy that examines the internal processes of the agent itself. This can be used for a variety of effects that include completing gaps in its own knowledge, correcting mistakes that may have occurred during the learning or reasoning processes, and determining what strategies to use for learning and reasoning. Learning by correcting mistakes is the process of agent learning information through isolating and examining an error, then using the methods of generalization or specialization to create a better model of the concept, preventing the error from recurring in the future. These two concepts are related in that Metareasoning can be seen as encompassing the concept of correcting mistakes.

Part C:

- **C1:**

I would use a combination of frames and configuration in order to solve this problem. My system would work by initially creating a frame for every bottle on the table. Each frame would hold the various attributes of the bottle, such as height, position, and neighbors, as slots within itself. Then my system would initialize the set of labels that it will be trying to apply to the bottles. These labels represent what the liquid inside the bottle could be. In the case of this problem the number of labels is limited to 3x wine, 3x poison, 1x forward, 1 backwards. Finally, my configuration system applies the labels depending on any existing rules that would have been provided by a user. As a supporting technique, my system would have to be equipped with the ability to convert a user's textually provided rules into a more machine understandable format. In the case of this problem my system would take the rules like "Neither dwarf nor giant holds death in their insides" and converts them into a format like: Bottle(Max(Size)) = Safe, Bottle(Max(Size)) = Safe.

- **C2:**
  Below is a diagram that represents my representation. Looking at it you can see that I represent each bottle with a frame. Each of these frames has the slots position, Left Neighbor, Right Neighbor, and Size. The Size of the bottle is a number that represents its size compared to all of its peers, a size of 1 is the smallest, and a size of 7 would be the largest. I also have a set of labels, because this problem limits them to a finite amount, we only have to worry correctly assigning 3x wine, 3x poison, 1x forward, 1 backwards label. Shown below Is also the list of requirements that would be fed to my agent from reading the puzzle itself. These restrictions not only define which bottles the labels can be assigned to, they also define the characteristics of the labels themselves. For example, you can observe that Poison has been given the attribute “Dangerous” and “Detrimental” while wine was given the properties “Safe” and “Beneficial”.



Bottle 1 Frame

Position: 1
Left Neighbor: None
Right Neighbor: 2
Size: 2

Possible Label: %Conf

Bottle 2 Frame

Position: 2
Left Neighbor: 1
Right Neighbor: 3
Size: 7

Possible Label: %Conf

Bottle 3 Frame

Position: 3
Left Neighbor: 2
Right Neighbor: 4
Size: 1

Possible Label: %Conf

Bottle 4 Frame

Position: 4
Left Neighbor: 3
Right Neighbor: 5
Size: 6

Possible Label: %Conf

Bottle 5 Frame

Position: 5
Left Neighbor: 4
Right Neighbor: 6
Size: 5

Possible Label: %Conf

Bottle 6 Frame

Position: 6
Left Neighbor: 5
Right Neighbor: 7
Size: 4

Possible Label: %Conf

Bottle 1 Frame

Position: 7
Left Neighbor: 6
Right Neighbor: None
Size: 3

Possible Label: %Conf

Set of Possible Labels

Poison
Poison
Poison
Wine
Wine
Wine
Backwards
Fowards

Set of Requirements

Label(Poison) = Dangerous
Label(Poison) ≠ Safe
Label(Poison) = Detrimental
Label(Poison) ≠ Beneficial
Label(Wine) ≠ Dangerous
Label(Wine) = Safe
Label(Wine) ≠ Detrimental
Label(Wine) = Beneficial
Label(Backwards) ≠ Dangerous
Label(Backwards) = Safe
Label(Backwards) = Detrimental
Label(Backwards) ≠ Beneficial
Label(Fowards) ≠ Dangerous
Label(Fowards) = Safe
Label(Fowards) ≠ Detrimental
Label(Fowards) = Beneficial

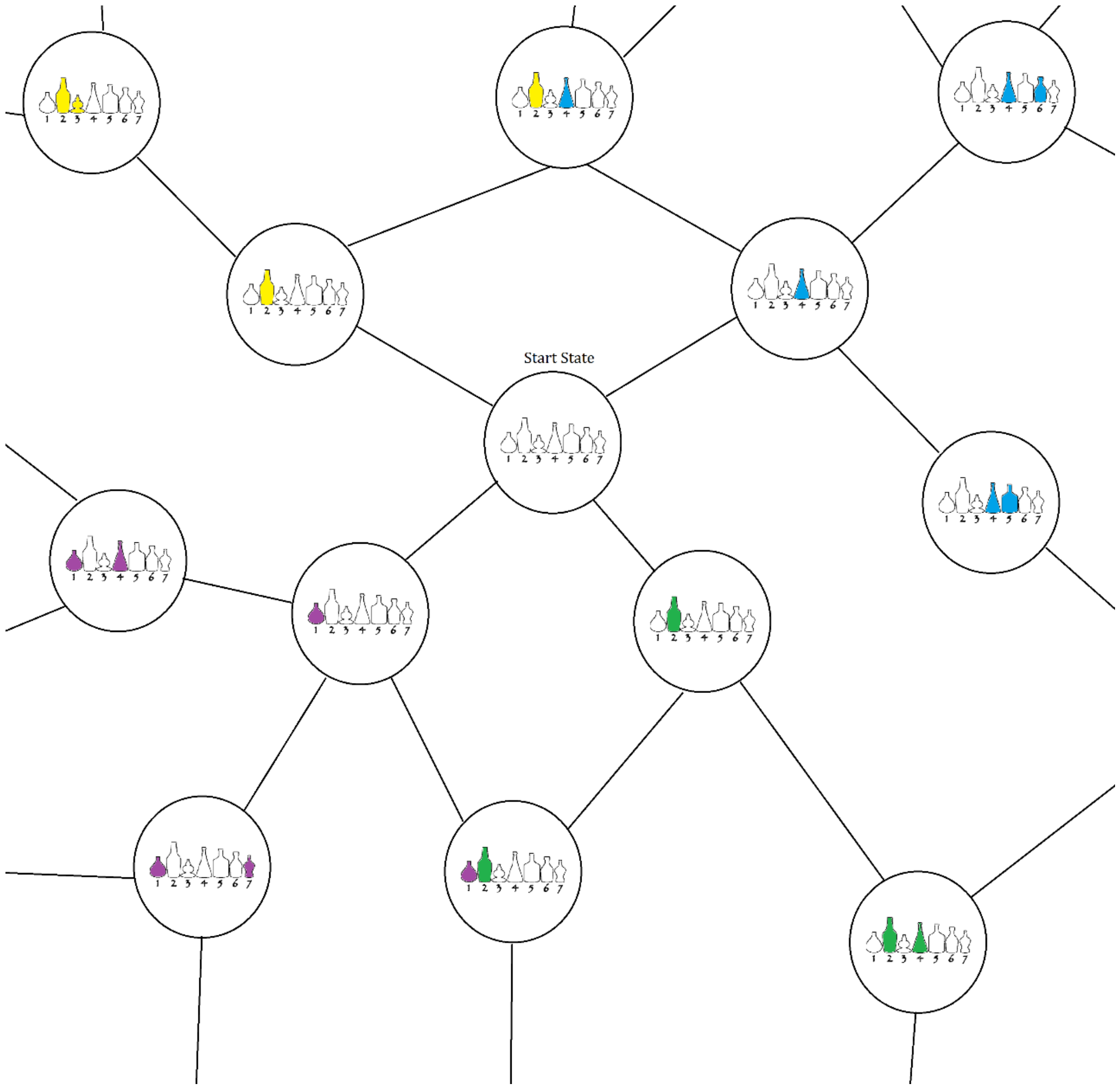
Bottle(Position),Label(Poison) < Bottle(Position),Label(Wine)
Bottle(Position == 1) = Detrimental
Bottle(Position == 7) = Detrimental
Bottle(Max(Size)) = Safe
Bottle(Min(Size)) = Safe
Bottle(Position == 2) = Bottle(Position == 6)

- **C3:**

My technique would first require some background domain knowledge about the various liquids that could fill the various bottles, in order to properly create their labels and attributes. For example, it would require background knowledge so that my agent can identify nettle wine as a harmless substance. After possessing such background knowledge, my agent would then create the various labels that correspond with the various liquids that could be assigned to the bottles. In this particular example, there are a finite number of labels allowed, so I can similarly limit the quantity of the labels in my representation. Now I read through and create a frame for each and every bottle in the problem. These frames will have slots that contain the relevant attribute values for that bottle, in the case of this problem, we only need the attributes Position, Left Neighbor, Right Neighbor, and Size. Now we read through the list of "hints" given in the riddle and convert them into usable requirement/restriction that we can then apply to my agent. My agent will use these requirements when performing the configuration steps. Without any restrictions my agent would likely simply assign its labels in a random formation. but upon applying these rules the possible formations are pruned until only the single correct solution remains. My agent recognizes when it hit some sort end state by running several simulations simultaneously and comparing their labeling. If the labeling differs, then that means there is some unknowns causing randomness in the results. If the solutions all match, then the agent knows that it has reached the final state possible from the hints provided and returns that as the solution.

- **C4:**

Another system that could potentially be used would be a graphical representation of the possible states of the potions. Then beginning at a starting position, the agent would search through the graph using its system of rules to avoid traveling down branches that lead to impossible situations. If the rules provided enough detail to reach a conclusive answer the searching agent should similarly reach an end goal, which would be the correct answer. Shown below is a depiction of a small section of what the graph could look like. there is a central node with all the bottles marked empty. Every edge is the equivalent of "one instance", where a single label is added, removed, or changed between the parent and receiving node.



- **C5:**

A metacognition system may compare the potential time and special resources required by each technique. This can be particularly useful if the number of bottles, attributes, or rules in the situation were changed. For example, if the problems had a small number(1-3) of bottles/attributes the graphical technique described in C4 performs faster than my C1 technique. However, as the number of bottles/variables increase my graphical search method would take exponentially more space and time to process, making the C1 method preferable to use.