Brad Ware
bware8@gatech.edu
Assignment 6

Using Logic to solve Raven's progressive matrices

Logic is a powerful representation of knowledge for any agent, because it allows one to evaluate statements into a true/false form. Logic has a formal notation that consists of predicates, which allow one to map objects into true/false conclusions. Predicates look like functions and take in the object as the argument. For example, if I was going to represent myself in formal logic, I might use some of the following predicates:

Male(Brad) ∧ Caucasian(Brad) ∧ ¬Blonde(Brad) ∧ Tall(Brad) ∧ Handsome(Brad)

(The last one might be debatable). However these are all predicates that can be evaluated to true and false statements about the object, myself. They are connected by "and" statements, but also could be connected through "or" statements with this ∨ being the symbol. With these predicates, If-Then statements can be formulated such as:

If a person is a male, dark-haired, and tall, then it is Brad.

Male(person) ∧ ¬Blonde(person) ∧ Tall(person) ⟶ Brad(person).

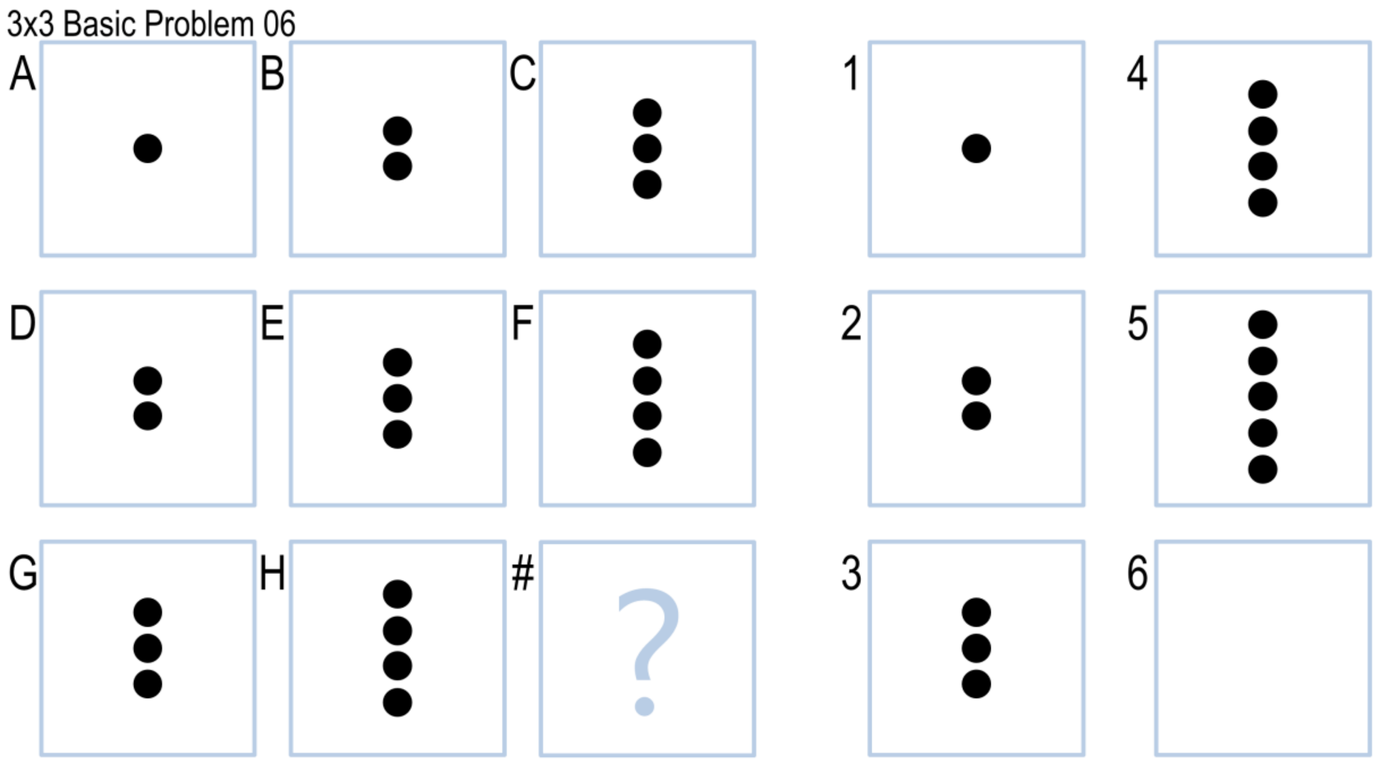
Here, logic is representing the first statement about identifying if someone is Brad. From there, we know rules of inference that allow us to draw conclusions even if we only have part of the information. Two of the more popular rules are Modus Ponens and Modus Tollens, where both are described below. Both of these are extremely useful when analyzing a situation and only having part of the story:

Modus Ponens: p ⟶ *q*
Known: p
∴ *q*

Modus Tollens: p ⟶ *q*
Known: ¬*q*
∴ ¬p

Both of these are extremely useful rules in the realm of logic that can be applied to numerous situations when trying to evaluate a series of predicates to true or false.

But how can logic be used to solve Ravens' progressive matrices? Well, more than likely we already use basic logic when solving these problems mentally. Even though we don't formally write out each predicate of every series of figures, we can computationally eliminate many of the answer choices purely from logic. For example, let's take a look at this problem:



In this problem, logic can easily used as a knowledge representation that could eliminate all of the other answer choices. For example, for figure C, some predicates could be:

Figure C
Circle(Object1) ∧ Circle(Object2) ∧ Circle(Object3) ∧ SameShape(FigureC) ∧ MoreObjectsThanPrev(FigureC) ∧ SameAngle(FigureC) …

More importantly, the logic representing the relationship between Figures A, B, and C could be represented as:

Group(FigureA, FigureB, FigureC)
SameShape(Group) ∧ ShapeCircle(Group) ∧ NumObjectIncreasing(Group) ∧ SameFill(Group) …

The logic statements for this Group could be combined with Groups DEF, ADG, and BEH. In this combination, only the predicates that were common between each group would be kept as predicate descriptors for the problem. So, this could be interpreted as compiling the intersection of all predicates between each group. From there, our agent could iterate through each of the answer choices and form a "dummy" group to evaluate the similarities like Group(FigureG, FigureH, AnsChoice1) etc. Another dummy group would also be made with figures C and F. Now, when evaluating the two dummy groups, we can compare it to the compiled If-Then statement from the intersections from each group's predicates. So for this problem, an intersection of predicates from groups ABC, DEF, ADG, and BEH may be:

SameShape(Group) ∧ ShapeCircle(Group) ∧ NumObjectIncreasing(Group) ∧ SameFill(Group) …

If a Group shares the same object, and each figure only contains Circles, and from each figure the number of objects increases, and every object in each figure is filled, then the answer choice used in the Group is correct.

SameShape(TempGroup) ∧ ShapeCircle(TempGroup) ∧ NumObjectIncreasing(TempGroup) ∧ SameFill(TempGroup) ⟶ Answer(AnsChoice)

where TempGroup is first G,H,AnsChoice and then C,F,AnsChoice. Doing this, our logic will return Figure 5 because it is the only answer choice that complies to all of the requirements "and-ed" together. None of the other shapes allow for each row and column to increase the number of objects as you iterate across and down. Even though this is a rather simple example, logic predicates can be combined in multiple ways and form many If-Then statements that can easily eliminate many of the answer choices for a Raven's progressive matrices problem. Because logic statements can quantifiably be simplified to true or false, this makes it much easier for our agent to process and categorize when trying to determine the correct answer choice.