Assignment 4: A Knowledge Base for Block World

In this assignment, you are going to add in the ability to not only add facts and draw inferences from them, but retract them and any inferences that they support. This is support a Blocks World Planner we will be building as the next step in this process.

Your knowledge base (hence KB) will include two sorts of elements: facts and rules.

The facts will be statements that includes predicates (e.g., Color, Size, Inst) that relate objects together. For example:

Block1 is an instance of a rectangle.

Inst(block1, rectangle)

Block1 is red

Color(block1, red)

Block1 is large

Size(block1, large)

Rectangles are blocks

Isa(rectangle, block)

You will also have rules. Rules will have a left hand side (LHS) and a right hand side (RHS).

In our system, the LHS will always be a conjunct of patterns that look exactly like facts that include variables so that they can match against any number of specific facts in our KB.

The RHS will always be a single pattern that is of the same predicate/argument form as facts in our KB. These patterns will reference the same arguments that are included in the LHS.

When all of the patterns in the LHS are true (that is there are facts in the data that match them with a consistent set of variable bindings) the variables in the RHS are replaced with the objects that matched the elements in the LHS and the new fact is added to the KB.

More technically, the new fact is instantiated using the bindings from the match.

So we might have a rule that states that something being on top of another thing means that the second is covered that would look like this:

(On(?x, ?y)) => Covered(?y)

The pattern on LHS is in a list because it will always be a conjunct and all of the arguments in both the LHS and the RHS are variables.

If we already know:

On(A, B)

This means that we can bind ?x to A and ?y B and we can replace ?y with B in the RHS and infer:

Covered(B)

And add it to our KB.

Of course, if we also know:

On(D,E)

We will also infer:

Covered(E)

So we are going to want to hold onto multiple sets of bindings. These might look like:

((?x: A, ?y: B), (?x: D, ?y: E))

Whenever we add a new fact to the KB, we check to see if it triggers a rule. And whenever we add a new rule, we check to see if it is triggered by existing facts.

As we add new rules with multiple statements to be checked in their LHS, however, we do not want to constantly check each and ever one of those elements whenever a new fact is added to our KB. In order to avoid this, we are going to use tiny trick.

Whenever a new rule or fact is added, we will only check the first element in the LHS of that rule (or multiple rules, if we have added a fact) against our data. If the first element of a rule matches, we will actually a new rule with the bindings associated with that match in place.

So imagine a rule like the one above, but with the additional constraint that a thing is only Covered if the object on top of it is the same size or bigger. Let's call that predicate "AtLeast." So our rule would look like:

(AtLeast(?x, ?y),On(?x, ?y))=> Covered(?y)

And we know that A is bigger than B (and that in turn it is AtLeast with regard to B).

AtLeast(A, B)

This means that the rule matches with:

((?x: A, ?y: B))

Because of this, we will infer a new rule that includes this binding:

((On(A, B))=> Covered(B)

If A is on B, then the result of adding this rule will trigger the actual inference. But if it isn't true, then the new rule just waits until someone asserts that A is now on B.

The idea here is that we will not only infer facts about object, but we will also infer new rules.

The assignment will be the construction of a KB to which you can add facts and rules and draw inferences based on their interaction.

- You will need to be able to Assert new facts and rules.
- Whenever you add a fact, you will need to check to see if it triggers are rule and, if so, make the appropriate inference of either a new fact or new rule. You will then need to Assert the Instantiated inference thus adding it to the KB.

A Tip: If you already have a fact or rule in you KB, you don't want to add it again bu tyou do want to add new supports for it.

Of course, we are looking to change facts through our actions. Our actions will sometimes add new facts but will also remove them. If we move A from B to C, we need to remove the fact that A is on B from our KB. In doing so, we also need to remove any inferences that we made earlier based on adding On(A, B) to our KB.

To do this, you will need to keep track of the inferences that you made based on the Assertion of new facts or rules so you can remove them from the KB as needed. Which is to say, you need to be able to Retract facts and rules as well as Assert them.

We also want our rules to manage not just adding, but also retracting facts. In particular, we want rules that remove facts from the KB that are contradicted by new facts. And, if we retract the new facts, we want to be able to re-Assert the facts that we removed.

Also remember that we are going to use this KB later to support our planner, so you need to be able to Ask it about what is true or not true. Given that you might be wondering what blocks are blue or clear and their may be multiple answers, Ask should be return a list of bindings lists that represent those multiple answers. So if we are wondering what things are blue:

Color(?x, blue)

And both blocks A and B are blue, Ask needs to be able to return this:

((?x: A), (?x: B))

Which essentially means:

"?X is blue" is true when A is bound to ?X or when B is bound to ?X. In other words, A and B are both true.

This gives us these requirements as well:

- You need to be able to Retract a fact from your KB.
- You need to be able to identify the facts and/or rules that this facts supports are Retract them as well.

- You will also need to have specific rules that Retract rather than Assert This will allow you to infer that something is no longer clear once it is covered.
- You also need to be able to Ask you KB about what is true and get back list of bindings lists (representing the different circumstances under which the statements you are wondering about are true).

So, you need to implements Assert, Retract and Ask.

- Assert adds facts and rules to the KB and infers new facts and rules as appropriate.
- Retract removes facts and rules from the KB and then retracts any existing fact and rules that they support.
- Ask, given a list of patterns (statements that may or may not have variables in them), returns the bindings in the KB that hold when those patterns are true.

You will need to build up facts and rules for the following statements that define a simple blocks world:

Cubes, **Rectangles**, **Pyramids** and **Spheres** are all examples of **Blocks**. That is to say, a **Cube** Isa **Block**.

Rectangles are bigger than **Cubes**.

A **Box** is a **Container**.

A **Table** is a **Surface**. **Tables** are always clear.

Cubes and **Rectangles** are also **Surfaces**.

There are four instances each of **Cubes**, **Rectangles**, **Pyramids** and **Spheres**. That is to say, if you had a **Cube** called **cube1**, it would be an Inst of a **Cube**. Of each of the four instances of each object, one is **red**, two are **blue** and one is **green**.

You have two boxes, one is **big** and **red** and one is **small** and **blue**.

You have one table.

Everything starts off on the table.

Everything is Clear.

And then you have some rules:

1. If a thing is a **Block** and is Clear, you can pick it up.

Note: An easier way to phrase this is: “You can pick up blocks that are clear.” Slightly more formally:

(Inst(?x, Block), Clear(?x)) => Liftable(?x)

1. You can put things on **Surfaces**, if they are Clear.

2. If you are an Instance of a thing and it Isa different thing, then you are an instance of that thing too.

Note: Again, slightly more formally:

(Inst(?x, ?y), Isa(?y, ?z)) => Inst(?x, ?z)

This will allow the system to infer that all of the specific **Blocks** are **Surfaces**.

1. If you put something in a small box, it is full.
2. If something is bigger than you, you are smaller than it.
3. If something is on top of you, you are covered.
4. If you are covered, then you are no longer (and must Retract) Clear.

Implementation

You will need to implement:

Assert: This takes a knowledge base (KB) and a new fact (or rule) and adds it to the KB. It also tests to see if any new facts or rules can be inferred.

Ask: This takes a KB and a fact and returns the lists of bindings lists that hold if he fact is true in the KB.

Retract: This takes a KB and a fact and removes that fact from the KB. If the fact supports other facts, they need to be retracted as well.

Match: Given two facts, it returns the list of bindings that hold if the facts match.

Instantiate: Given a fact and a list of bindings, this returns the facts with its variables bound to the values in the bindings list.

Infer: Given a rule and a fact, this returns the fact that can be inferred from those two. This might be a new rule where the left-hand side has been reduced and the variables are now bound based on the match against a component.

Testing:

You will test this against a file of statements to be found in the folder for this assignment. This is statements.txt. You should be able to read this file then update your KB and test against it.

Support:

There is a file read.py in the folder as well that will read in and parse new facts and rules for you. This is there so you don’t have to build low level files readers.