Project 2
CMSC421: Intro to Artificial Intelligence

Due April 15^{th} 2019

1 Introduction

You will be tasked with implementing an "advice taker" agent that can learn from facts provided to it, detect contradictions, and answer queries. These facts will come from an advice giver (e.g. you) who tells the program certain facts about the world. For instance, phrases such as "A dog is an animal." and "An animal is a thing." are facts you could tell the program and it could use those to infer that a *dog* is indeed a *thing* as well.

NOTE: No modules can be used ***except*** for the ones already provided within the code. If you decide that there is a module you think is useful *and* ok to use (e.g. doesn't implement the project for you), then **ASK** via Piazza if said module is ok so that a response can be made to the entire class.

You are to use any python environment that works with the project! Python 3.5 or greater.

2 Input

There are three types of phrases that your agent will encounter, which are as follows:

- **(A/An) <X> is (not) (a/an) <Y>.** ⇒ This phrase tells your agent a/an X (dog, cat, boat, name, made up word, proper name etc.) is also a/an Y (dog, cat, boat, name, made up word, etc...but NOT a proper name for the second noun). After receiving this input, your agent should be able to make the inference that if something is a X then it is a Y. Alternatively, if "not" is used, then your agent should recognize that a X is not a Y, if asked.
- **Is (a/an) <X> (a/an) <Y>?** ⇒ This phrase is a question to your agent on whether or not X is a Y, if it knows at all.
- **<anything else>** ⇒ If an input is provided that doesn't match the above phrases, said input is considered invalid.

You will be provided with a parser class that will parse the input as either a fact input, a query input, or an invalid input.

3 Responses

Despite only encountering two phrases (and a third invalid phrase type), there are a variety of responses that your agent will provide.

- **OK** ⇒ The 'OK' responses should be given after your agent has received a new fact that is neither known already nor a contradiction.
- **IK** ⇒ The 'IK' response should be given if your agent is told a fact that it already knows.
- **Y** ⇒ The 'Y' response should be given when your agent is asked a query and the answer within the agent's knowledge base is yes.
- **N** ⇒ The 'N' response should be given when your agent is asked a query and the answer within the agent's knowledge base is no.

- **IDK** ⇒ The 'IDK' response should be given when your agent is asked a query and the agent does not know whether the answer is yes or no.
- **CONTR** ⇒ The 'CONTR' response should be given when a fact is input that contradicts one or more facts already present within your agent's knowledge base.
- **SMT** ⇒ The 'SMT' response should be given when your agent is asked a query and the answer within the agent's knowledge base indicates that the query MAY be true sometimes, but is not guaranteed to be true all the time.
- **INV** ⇒ The 'INV' response should be given if the input provided is invalid.

In order to be able to swap out different "flavors" of responses, one of the parameters passed into your agent will be a response dictionary, wherein the keys are the bold strings given above. For instance, to get the 'yes' response, one would use the following syntax with your agent's code:

self.responses['Y']

A default response dictionary will be used if no response file is provided.

4 Knowledge Base Behavior

The facts in your agent's knowledge base are transitive, and thus if a new fact is added, it will gain any knowledge that links to it. For example, if "A dog is a canine." and "A canine is an animal." are facts entered into your agent's knowledge base, then the fact "A dog is an animal." when input should cause the agent to provide the 'IK' response to indicates that it already knows this fact. In a similar vein, if the query "Is a dog an animal?" is asked, then your agent should output the 'Y' response to indicate that yes it is.

Elaborating on the query mentioned above, "Is a dog an animal?" should return yes, BUT the query "Is an animal a dog?" should return the response 'SMT' to indicate that sometimes an animal is a dog, but not all the time. And, considering the example facts mentioned, if a query is asked such as "Is a dog a bird?", your agent's response should be 'IDK' to indicate that your agent does not know whether this is true or not or even sometimes true. *YOU* may know the answer is no, but your agent in this example has yet to learn that fact.

The "not" facts are also transitive. So if "A dog is not a cat." is an entered fact, then anything that is a dog should also not be considered a cat. For instance, if the fact "A poodle is a dog." is entered along with the previous fact, then the query "Is a poodle a cat?" or "Is a cat a poodle?" should both return the 'N' response to indicate that no they are not. Be careful in handling both regular and 'not' facts, as adding 'not' facts into the agent can make things a bit more complicated to keep straight.

Furthermore, attempting to add a new fact to your agent's knowledge base could result in a contradiction with a fact (or 'not' fact) that already exists. For instance, in the above example if the facts "A dog is not a cat." and "A poodle is a dog." are added to your agent's knowledge base, attempting to add the fact "A poodle is a cat." or "A cat is a poodle." should both result in a contradiction and thus return the 'CONTR' response to indicate such. The new contradicting fact should also NOT be added to your agent's knowledge base. Facts that will create cycles should also cause a contradiction response and not be included in the knowledge base.

Lastly, proper names such as 'Fido' can be included as the first noun in a fact (e.g. 'Fido is a dog.') However proper names will not be included as the second noun in a fact. However, proper nouns can be included as both the first or second noun in a query (e.g. 'Is Fido a dog?' and 'Is a dog Fido?') The proper name 'Fido' should be treated as different from the noun 'fido', given that facts are entered that use both. You won't be able to rely on capitalization, but a proper noun (at least in testing) will not have a 'a/an' in front of it in a fact or query.

5 Code

You will be provided with three main files and a directory: ***kb.py, parser_proj2.py, shell.py, all_tests***.

The ***kb.py*** file will contain your agent's code. You will be tasked with filling in the requisite functions (adding helper functions if you find it necessary) and setting up the requisite data structures to handle the knowledge base behavior indicated above. It is *strongly* encouraged that you use a graph or graph-like

structures to keep track of your knowledge base, as this will allow you to use search functions (which you would have to create yourself), among other benefits.

The ***parser_proj2.py*** file contains a parser class with regular expressions for the input facts and queries. DO NOT change this file.

The ***shell.py*** file runs a simple shell class that will allow you to input facts and/or queries. DO NOT change this file.

all_tests is a directory containing a large number of provided public tests that will automatically run your knowledge base code and compare the results to the desired output.

In order to run your agent, you will execute the command:

python3 shell.py

This will bring up a simple shell that will request input facts and/or queries that you will enter manually. Said shell will also print out the responses of your agent after each input.

The ***shell.py*** script also has options to make testing your agent easier. See the following command, which indicates all the available optional arguments:

python3 shell.py --input=<input_file_name> --prior_kb=<prior_kb_file_name>
--save_kb=<kb_save_file_name> --output=<output_file_name>
--responses=<response_file_name>

An example of the shell script with all the options is as follows: *"python3 shell.py --input=my_input.txt --prior_kb=old_kb.txt --save_kb=save_kb_here.txt --output=my_output.txt --responses=resp.txt"*.

The ***input*** option will take a text file input of facts and queries so that you don't have to manually enter them (note that each fact/query needs to be on its own line). This will then skip the manual entering phase but will still provide the same output that the manual entering phase would.

The ***prior_kb*** option will load facts from a prior run of your agent into the current run of your agent.

The ***save_kb*** option will save the facts that your agent learns to a text file such that said facts can be loaded into a new instance of your agent at some later point in time (via the prior_kb option).

The ***output*** option will take a file name and save the output from your agent to said file name.

The ***responses*** option will load in alternative responses for the requisite response types. Not all response types need to be represented in this file, as any that aren't will keep the default response.

NOTE: The default kb file will always output the 'INV' answer before you implement anything.

6 Testing

Your code will be tested on the output it provides to given input facts and queries. Public tests will be provided so that you can test your agent implementation, though it is recommend that you make your own tests as well. In order to run the public tests, go into the ***all_tests*** directory and enter the following command in your terminal:

python3 all_tests.py

NOTE: These tests aren't guaranteed to be exhaustive!

7 Deliverables

Please submit your ***kb.py*** file to elms.

At the top of your project, please put a comment giving your full name, UID, and directory id.

8 Hints

We'd recommend a graph-like structure for storing your knowledge base, as it'll allow search algorithms (that you'd have to implement!) to be used, such as breath first search. This would help you to store data efficiently and still maintain connections between nodes. In a similar vein, it might be a good idea to keep a separate data structure for "not" facts.

For testing on your own, try to create natural trees of things, such as "A dog is an animal.", "An animal is a thing." etc, and then thinking of what queries on this would be in reality. "Is a dog an animal?" should be yes and "Is an animal a dog?" should be sometimes, as sometimes animals are indeed dogs, but not all the time. Try adding facts in different orders too, to make sure you get the same results. Try adding cycles and facts that would cause contradictions and try these in a variety of input combinations, as your implementation might only work in a narrow way (e.g. "A dog is not a cat." and then asking "Is a cat a dog?" and "Is a dog a cat?" should both return the "no" answer.)

As for proper nouns, don't rely on input punctuation to determine what is a proper noun or not, but instead look at the difference between "a fido is a dog." and "fido is a dog." Look at the parser output in these two situations to help. When storing "fido" in your knowledge base, if you've determined it's a proper noun you would want to somehow indicate it as such to distinguish it from "fido" that is a common noun (not a proper noun).

And **DON'T FORGET** to make your own test cases!