COGS 125 / CSE 175
Introduction to Artificial Intelligence
Specification for Programming Assignment #2

David C. Noelle

Due: 11:59 P.M. on Friday, November 5, 2021

Overview

This programming assignment has two main learning goals. First, the assignment will provide you with an example of an automated reasoning inference procedure for first-order logic implemented in Python. You will be able to explore the use of such an inference procedure by populating the knowledge base with various sentences. Second, and more prominently, this assignment will give you practice translating English sentences into first-order logic definite clauses. You will produce a knowledge base of facts and rules that communicate some general knowledge that has been provided to you, and you will provide these sentences in a form that they may be directly used by the provided Python inference engine.

Submission for Evaluation

To complete this assignment, you must generate one Python script file, called "`knowledge.py`". That file is to contain a global variable, called `monster_sentences`, initialized to contain a list of strings, with each string encoding a sentence in first-order logic. This is the only file that you should submit for evaluation.

To submit your completed assignment for evaluation, log onto the class site on CatCourses and navigate to the "Assignments" section. Then, locate "Programming Assignment #2" and select the option to submit your solution for this assignment. Provide your single program file as an attachement. Do not upload any other files as part of this submission. Comments to the teaching team should appear as header comments in your Python source code file.

Submissions must arrive by 11:59 P.M. on Friday, November 5th. Please allow time for potential system slowness immediately prior to this deadline. You may submit assignment solutions multiple times, and only the most recently submitted version will be evaluated. As discussed in the course syllabus, late assignments will not be evaluated and will receive *no credit*.

If your last submission for this assignment arrives by 11:59 P.M. on Tuesday, November 2nd, you will receive a 10% bonus to the score that you receive for this assignment. This bonus is intended to encourage you to try to complete this assignment early.

Activities

Domain

You are to provide a Python script file that defines the global variable `monster_sentences` and initializes it to a list of strings, with each string being a first-order logic sentence. These sentences are to capture the domain knowledge provided below.

The domain of this knowledge base is a fictional murder that took place in the cemetery last night. Somebody has been killed, and the possible villians include vampires, werewolves, and witches. You will be writing first-order logic sentences that reflect general knowledge about these monsters and the terrible things that they do to their victims. To test your knowledge base, the details of a specific homicide will be encoded as additional logical sentences, and the inference engine will be asked if the identity of the murderer (or murderers) can be deduced from the provided knowledge. If you translate the given English sentences correctly, the culprit will be caught (at least in some cases).

Knowledge

You are to translate the following statements into first-order logic definite clauses:

- Celene is a vampire.
- Mario is a werewolf.
- Mildred is a witch.
- Bob, Lin, and Maurice are people.
- A person who is dead is a victim.
- All vampires, werewolves, and witches are monsters.
- Any monster present at the crime scene is a suspect.
- If a vampire is suspected and the victim was bitten, then the vampire killed the victim.
- If a werewolf is suspected and the victim was eaten, then the werewolf killed the victim.
- If a witch is suspected and the victim was poisoned, then the witch killed the victim.
- If a victim is drained of blood but has an intact body, then the victim was bitten.

- If a victim is drained of blood and has body pieces missing (i.e., is incomplete), then the victim was eaten.
- If a victim's body is intact, and the victim's skin complexion is green, blue, or purple, or if it is pale with boils, then the victim was poisoned.
- If a victim's skin complexion is pale and the body is either cold or punctured, then the victim is drained of blood.
- If a victim's body is complete (i.e., not incomplete), then the victim is not dismembered or disemboweled.

These English sentences contain all of the knowledge to be encoded in the knowledge base. Additional knowledge should not appear in your list of logical sentences.

Ontology

The first-order logic sentences to be produced must use the following ontology of constants and predicate symbols:

- *Bob*, *Celene*, *Lin*, *Mario*, *Maurice*, *Mildred* : constant symbols referring to individuals
- *Pale*, *Blue*, *Green*, *Purple* : constant symbols for skin complexion colors
- *Bitten(x)* : true iff *x* has been bitten
- *Boils(x)* : true iff the skin of *x* has boils
- *Cold(x)* : true iff the body of *x* is cold to the touch
- *Complexion(x, y)* : true iff the skin of *x* has the complexion color *y*
- *Dead(x)* : true iff *x* is dead
- *Disemboweled(x)* : true iff the body of *x* is missing internal organs
- *Dismembered(x)* : true iff the body of *x* is missing limbs
- *Drained(x)* : true iff the body of *x* has been drained of blood
- *Eaten(x)* : true iff *x* has been eaten
- *Incomplete(x)* : true iff the body of *x* is missing parts
- *Intact(x)* : true iff the body of *x* is relatively whole and intact
- *Killed(x, y)* : true iff *x* murdered *y*
- *Monster(x)* : true iff *x* is a monster

- *Person(x)* : true iff *x* is a person
- *Poisoned(x)* : true iff *x* has been poisoned
- *Present(x)* : true iff *x* was present at the crime scene
- *Punctured(x)* : true iff the body of *x* has puncture wounds
- *Suspect(x)* : true iff *x* is a murder suspect
- *Vampire(x)* : true iff *x* is a vampire
- *Victim(x)* : true iff *x* was the victim of a murder
- *Werewolf(x)* : true iff *x* is a werewolf
- *Witch(x)* : true iff *x* is a witch

In the descriptions of these predicates, "iff" is shorthand for "if and only if". Note that capitalization matters. Constants and predicate symbols must start with a capital letter. Variables are all lowercase. You should not introduce any other constants or predicates (or function symbols) beyond those explicitly listed in this ontology.

Translation

The knowledge provided as English sentences is to be translated into a list of first-order logic sentences. These will be gathered, along with sentences describing the specific murder case, in order to produce a Python object of the class `FolKB` (first-order logic knowledge base).

Note that every sentence *must* be a definite clause. The inference engine that you are given implements a backward chaining algorithm that only works on knowledge bases consisting of definite clauses. Other kinds of first-order logic sentences are not allowed.

Also note that the number of first-order logic sentences needed to encode the provided knowledge need not be the same as the number of presented English sentences. In particular, it is not unusual for a single English sentence to be translated into multiple first-order logic definite clauses.

A template of the "`knowledge.py`" file is provided for you. Your only task is to set the value of the `monster_sentences` variable to a list of your definite clauses, with each sentence given as a string. In these strings, the conjunction operator is the ampersand ("&"), and the disjunction operator is the vertical bar ("|"). The implication operator is two equal signs followed by a greater-than sign ("==>"). Example sentences appear in the template script file.

Testing

The given "`main.py`" Python script provides a driver for testing the program. This file defines a specific murder case as a collection of definite clauses. These are concatenated to the `monster_sentences` list, and the result is used to create a `FolKB` object. This knowledge base is then passed to the inference engine's ASK function ("`fol_bc_ask`"), along with a query

concerning acts of murder: "*Killed(killer, victim)*". If this query can be proven from the knowledge base for some values of the variable *killer* and the variable *victim*, the inference procedure will output the values of these variables that allow the *Killed* query to be inferred.

You can test your list of sentences by temporarily modifying "`main.py`" to specify different situations in the global variable `case_sentences`. Different cases will exercise your first-order logic sentences in different ways, producing different results. Once you submit your solution, the teaching team will test it by examining the results from various murder cases.

The code for the backward chaining inference engine is contained in the file "`logic.py`". This Python script makes use of classes and functions in the script file "`utils.py`". These two files, along with "`knowledge.py`" and "`main.py`", are needed to execute the inference procedure, testing your solution. With the exception of initializing `monster_sentences`, *your solution must work without modifying these files in any way*. These files are provided to you in a ZIP archive file called "`PA2.zip`", which is available in the "Assignments" section of the class CatCourses site, under "Programming Assignment #2". The contents of these Python code files will be briefly discussed during a laboratory session, and comments in these files should assist in your understanding of the provided code. Questions are welcome, however, and they should be directed to the teaching team.

While a single test case appears in the "`main.py`" script, this case is insufficient to fully test your knowledge base. It is possible for your solution to contain serious errors but still perform well on this example. Thus, part of this assignment includes coming up with as many good test cases as possible and ensuring that your solution produces appropriate output for all of them. Your test cases will not be submitted for evaluation, but the quality of your solution will likely depend on the breadth of testing that you perform.

Your submission will be evaluated for accuracy. Note that, as discussed in the course syllabus, submissions that fail to run without crashing on the laboratory PyCharm IDE will not be evaluated and will receive *no credit*.

As for all assignments in this class, submitted solutions should reflect the understanding and effort of the individual student making the submission. Not a single line of computer code (or single first-order logic sentence) should be shared between course participants. If there is ever any doubt concerning the propriety of a given interaction, it is the student's responsibility to approach the instructor and clarify the situation *prior* to the submission of work results. Also, helpful conversations with fellow students, or any other person (including members of the teaching team), should be explicitly mentioned in submitted assignments (e.g., in comments in the submitted source code files). These comments should also explicitly mention any written resources, including online resources, that were used to complete the exercise. Citations should clearly identify the source of any help received (e.g., "Dr. David Noelle" instead of "a member of the teaching team", "The Python Tutorial at `docs.python.org/3/tutorial/`" instead of "Python documentation"). Failure to appropriately cite sources is called *plagiarism*, and it will not be tolerated! Policy specifies that detected acts of academic dishonesty must result minimally with a zero score on the assignment, and it may result in a failing grade in the class. Please see the course syllabus for details.

The members of the teaching team stand ready to help you with the learning process embodied by this assignment. Please do not hesitate to request their assistance.