CS50's Introduction to Artificial Intelligence with Python

cs50.harvard.edu/ai/2020/projects/1/knights

Knights

Write a program to solve logic puzzles.

When to Do It

By Monday, January 1, 2024 at 10:29 AM GMT+5:30.

How to Get Help

1. Ask questions via Ed!
2. Ask questions via any of CS50's communities!

Background

In 1978, logician Raymond Smullyan published "What is the name of this book?", a book of logical puzzles. Among the puzzles in the book were a class of puzzles that Smullyan called "Knights and Knaves" puzzles.

In a Knights and Knaves puzzle, the following information is given: Each character is either a knight or a knave. A knight will always tell the truth: if knight states a sentence, then that sentence is true. Conversely, a knave will always lie: if a knave states a sentence, then that sentence is false.

The objective of the puzzle is, given a set of sentences spoken by each of the characters, determine, for each character, whether that character is a knight or a knave.

For example, consider a simple puzzle with just a single character named A. A says "I am both a knight and a knave."

Logically, we might reason that if A were a knight, then that sentence would have to be true. But we know that the sentence cannot possibly be true, because A cannot be both a knight and a knave – we know that each character is either a knight or a knave, but not both. So, we could conclude, A must be a knave.

That puzzle was on the simpler side. With more characters and more sentences, the puzzles can get trickier! Your task in this problem is to determine how to represent these puzzles using propositional logic, such that an AI running a model-checking algorithm could solve these puzzles for us.

Getting Started

Download the distribution code from https://cdn.cs50.net/ai/2020/x/projects/1/knights.zip and unzip it.

Understanding

Take a look at `logic.py`, which you may recall from Lecture 1. No need to understand everything in this file, but notice that this file defines several classes for different types of logical connectives. These classes can be composed within each other, so an expression like `And(Not(A), Or(B, C))` represents the logical sentence stating that symbol `A` is not true, and that symbol `B` or symbol `C` is true (where "or" here refers to inclusive, not exclusive, or).

Recall that `logic.py` also contains a function `model_check`. `model_check` takes a knowledge base and a query. The knowledge base is a single logical sentence: if multiple logical sentences are known, they can be joined together in an `And` expression. `model_check` recursively considers all possible models, and returns `True` if the knowledge base entails the query, and returns `False` otherwise.

Now, take a look at `puzzle.py`. At the top, we've defined six propositional symbols. `AKnight`, for example, represents the sentence that "A is a knight," while `AKnave` represents the sentence that "A is a knave." We've similarly defined propositional symbols for characters B and C as well.

What follows are four different knowledge bases, `knowledge0`, `knowledge1`, `knowledge2`, and `knowledge3`, which will contain the knowledge needed to deduce the solutions to the upcoming Puzzles 0, 1, 2, and 3, respectively. Notice that, for now, each of these knowledge bases is empty. That's where you come in!

The `main` function of this `puzzle.py` loops over all puzzles, and uses model checking to compute, given the knowledge for that puzzle, whether each character is a knight or a knave, printing out any conclusions that the model checking algorithm is able to make.

Specification

An automated tool assists the staff in enforcing the constraints in the below specification. Your submission will fail if any of these are not handled properly, if you import modules other than those explicitly allowed, or if you modify functions other than as permitted.

Add knowledge to knowledge bases `knowledge0`, `knowledge1`, `knowledge2`, and `knowledge3` to solve the following puzzles.

- Puzzle 0 is the puzzle from the Background. It contains a single character, A.
  A says "I am both a knight and a knave."
- Puzzle 1 has two characters: A and B.
  - A says "We are both knaves."
  - B says nothing.

- Puzzle 2 has two characters: A and B.
  - A says “We are the same kind.”
  - B says “We are of different kinds.”
- Puzzle 3 has three characters: A, B, and C.
  - A says either “I am a knight.” or “I am a knave.”, but you don’t know which.
  - B says “A said ‘I am a knave.’”
  - B then says “C is a knave.”
  - C says “A is a knight.”

In each of the above puzzles, each character is either a knight or a knave. Every sentence spoken by a knight is true, and every sentence spoken by a knave is false.

Once you’ve completed the knowledge base for a problem, you should be able to run `python puzzle.py` to see the solution to the puzzle.

Hints

- For each knowledge base, you’ll likely want to encode two different types of information: (1) information about the structure of the problem itself (i.e., information given in the definition of a Knight and Knave puzzle), and (2) information about what the characters actually said.
- Consider what it means if a sentence is spoken by a character. Under what conditions is that sentence true? Under what conditions is that sentence false? How can you express that as a logical sentence?
- There are multiple possible knowledge bases for each puzzle that will compute the correct result. You should attempt to choose a knowledge base that offers the most direct translation of the information in the puzzle, rather than performing logical reasoning on your own. You should also consider what the most concise representation of the information in the puzzle would be.

  For instance, for Puzzle 0, setting `knowledge0 = AKnave` would result in correct output, since through our own reasoning we know A must be a knave. But doing so would be against the spirit of this problem: the goal is to have your AI do the reasoning for you.
- You should not need to (nor should you) modify `logic.py` at all to complete this problem.

How to Submit

You may not have your code in your `ai50/projects/2020/x/knights` branch nested within any further subdirectories (such as a subdirectory called `knights` or `project1a`). That is to say, if the staff attempts to access `https://github.com/me50/USERNAME/blob/ai50/projects/2020/x/knights/puzzle.py`, where `USERNAME` is your GitHub username, that is exactly where your file should live. If your file is not at that location when the staff attempts to grade, your submission will fail.

1. Visit this link, log in with your GitHub account, and click **Authorize cs50**. Then, check the box indicating that you'd like to grant course staff access to your submissions, and click **Join course**.
2. Install Git and, optionally, install `submit50`.
3. If you've installed `submit50`, execute

   ```
   submit50 ai50/projects/2020/x/knights
   ```

   Otherwise, using Git, push your work to `https://github.com/me50/USERNAME.git`, where `USERNAME` is your GitHub username, on a branch called `ai50/projects/2020/x/knights`.

4. Submit this form.

You can then go to https://cs50.me/cs50ai to view your current progress!