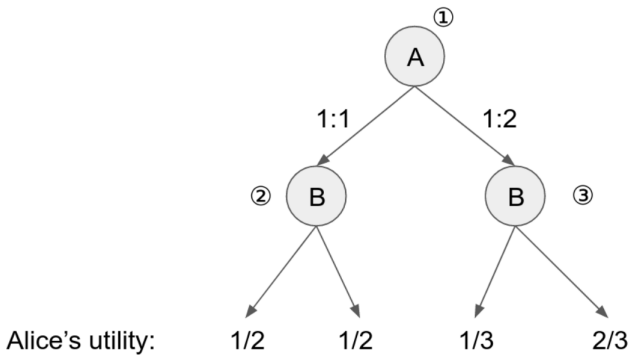
CSC 665: Artificial Intelligence

Homework 2

By turning in this assignment, I agree to abide by SFSU's academic integrity code and declare that all of my solutions are my own work.

1 Cake

Alice and Bob are sharing a cake. First Alice divides the cake into two pieces, then Bob chooses one piece, leaving Alice with the other piece. Alice can either divide the cake in half (a ratio of 1 : 1) or in a 1 : 2 ratio. Modeling this as a game gives the following game tree.



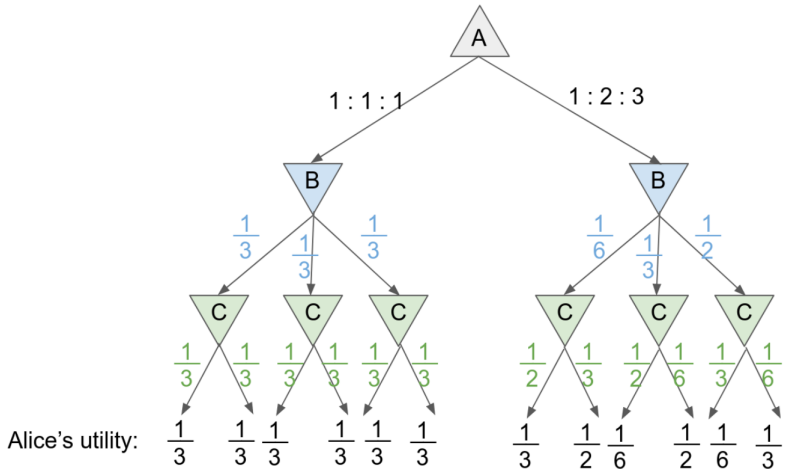
For each of the following scenarios, write down the value for each of the three non-terminal nodes of the game tree (labeled ①, ②, ③ above). In other words, write down the utility that Alice should expect to receive in each of the game states given knowledge of the player strategies described below.

a. (5 points) Suppose both Alice and Bob are playing adversarially, each trying to maximize the amount of cake they receive.

b. (5 points) Suppose Alice still tries to maximize her share of the cake, but now Bob plays collaboratively, working to also help Alice get as much of the cake as possible.

c. (10 points *extra credit*) Suppose Alice still tries to maximize her share of the cake, but now Bob plays randomly, choosing the larger piece of cake with probability p. The state values will now be expressions involving p. How should Alice divide the cake (i.e. what action should she take at the root node) according to the value of p?

Alice and Bob are joined by a 3rd friend, Carol. Now Alice divides the cake into three pieces, then Bob chooses one piece, after which Carol chooses one of the two other pieces, leaving Alice with the final remaining piece. Each player is trying to maximize their share of cake. To begin, Alice can either divide the cake into three equal pieces (a ratio of 1 : 1 : 1) or in a 1 : 2 : 3 ratio. Here is the game tree for this new scenario.



Note that the edges coming out of nodes B and C are labeled with the sizes of the pieces of cake that Bob and Carol choose, respectively.

d. (10 points) If we run alpha-beta pruning on this game tree, expanding nodes depth-first from left to right, which leaf nodes remain unvisited? (Note that by leaf node we mean the numbers indicating Alice's utility at the bottom of the tree.)

e. (5 points) Now suppose we swap the order of the two subtrees of the root node A and re-run alpha-beta pruning. Are the same leaf nodes pruned as in the previous part? If not, indicate which leaf nodes remain unvisited in this execution of the algorithm.

2 Logical formulas

In addition to the five connectives of propositional logic that we discussed in class, there are other connectives that are occasionally used. One common connective is known as "exclusive or" and is denoted by $\oplus$. The expression $P \oplus Q$ is true if and only if exactly one of P or Q is true.

a. (5 points) Write down an expression that is equivalent to $P \oplus Q$, using only the connectives $\wedge, \vee, \neg$.

Consider, in first-order logic, the following predicate definitions.

- Student(x) represents the predicate that x is a student.
- Course(x) represents the predicate that x is a course.
- Enrolled(x, y) represents the predicate that x is enrolled in y.

Suppose we also have the atomic symbols A and B for students Alice and Bob. For each natural language assertion below, write down an expression in first-order logic that encodes the assertion.

b. (5 points) "There is a course that Alice and Bob are both enrolled in."

c. (5 points) "Bob is enrolled in every course that Alice is enrolled in."

d. (5 points) "Anyone who is enrolled in a course is a student."

e. (5 points) "Every course has at least one student enrolled in it."

3 Knights and knaves

In 1978, logician Raymond Smullyan published *What is the name of this book?*, a book of logic puzzles. Among the puzzles in the book was 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 a 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 can conclude that A must be a knave.

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

In the `knights/` directory, take a look at `logic.py`. No need to understand everything in this file, but notice that we define several classes for different types of logical connectives. These classes can be composed with each other, so an expression like `And(Not(A), Or(B, C))` represents the logical formula $\neg A \wedge (B \vee C)$.

Observe that `logic.py` also contains a function `model_check`, which takes as input a knowledge base and a query. `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 `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.

Add knowledge to the knowledge bases to solve the following puzzles.

a. (5 points) Puzzle 0 is the puzzle from the example above. It contains a single character, A.
   - A says, "I am both a knight and a knave."

b. (5 points) Puzzle 1 has two characters: A and B.
   - A says, "We are both knaves."
   - B says nothing.

c. (5 points) Puzzle 2 has two characters: A and B.

- A says, "We are the same kind."
- B says, "We are different kinds."

d. (5 points) 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 of these sentences was said.
- B says, 'A said, "I am a knave."'
- B then says, "C is a knave."
- C says, "A is a knight."

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 Knights and Knaves 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 the inference algorithm do the reasoning for you.
- You should not need to (nor should you) modify `logic.py` to complete this problem.

4 Odds and evens

In this problem, we will see how to use logical methods to automatically prove mathematical theorems. We will focus on encoding the theorem and leave the proving part to the logical inference algorithm. Here is the theorem.

If the following antecedents hold:

a. (5 points) Each number x has exactly one successor, which is not equal to x.

b. (5 points) Each number is either odd or even, but not both.

c. (5 points) The successor of an even number is odd.

d. (5 points) The successor of an odd number is even.

e. (5 points) For every number x, the successor of x is larger than x.

f. (5 points) "Larger than" is a transitive relation: if x is larger than y and y is larger than z, then x is larger than z.

Then we have the following consequence: For each number, there is an even number larger than it.

Note: in this problem, "larger than" is just an arbitrary relation, and you should not assume it has any prior meaning. In other words, don't assume things like "a number can't be larger than itself" unless explicitly stated.

In `ints/submission.py`, fill out the `ints` function to construct 6 formulas — one for each of the theorem's above hypotheses. The consequence has been filled out for you as an example (`query` in the code). Note that variables must be represented with a string that starts with the `$` symbol, to distinguish them from atomic symbols and predicates. See `examples.py` for more examples of formulas using the syntax of this homework's first-order logic Python library.

For this problem, we have provided an autograder to evaluate the correctness of your implementation. You can test your code using the following commands:

```
python grader.py 4a-0
python grader.py 4a-1
python grader.py 4a-2
python grader.py 4a-3
python grader.py 4a-4
python grader.py 4a-5
python grader.py 4a-all  # Tests the conjunction of all the formulas
```

To finally prove the theorem, tell the `formulas` to the knowledge base and ask the `query` by running model-checking (on a finite model):

```
python grader.py 4a-run
```

You can run all these tests at once with

```
python grader.py
```

If you pass all the autograder tests, you can be fairly confident that you will receive full credit for this problem.

Submission

Submission is done on Canvas. You should submit three files: one containing your solutions to the written problems, and one for each of the two coding problems.

- Submit your written solutions in a single PDF file with your name at the top. Make sure to clearly indicate the number and letter of the problem corresponding to each solution. It is okay to hand-write your solutions and then scan them into a PDF, but *only if your handwriting is legible.*
- Submit your coding solutions in the provided `.py` files. For problem 3 you should submit `puzzle.py`, and for problem 4 you should submit `submission.py`. Do not modify the names of these files after downloading them from the course website.