B551 Assignment 4: Logic and CSPs

Spring 2015
Due: Friday November 6, 11:59PM
(You may submit up to 48 hours late for a 10% penalty.)

This assignment will give you a chance to practice probability problems, both on pen-and-paper and in practice.

For this assignment, unlike in some past assignments, **you are to work individually, not in a team**.

Academic integrity. You may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about Python syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials (e.g. in source code comments). However, the work and code that you submit must be your own work, which you personally designed and wrote. You may not share written answers or code with any other students, nor may you possess code or solutions written by another student, either in whole or in part, regardless of format.

Part 0: Getting started

We've created a github repo for you, as usual. To get started, clone the github repository:

`git clone https://github.iu.edu/cs-b551/`*your-user-name*`-a4`

Part 1: Written Problems

As usual, turn these in as PDFs via GitHub, either typed, or neatly handwritten and scanned.

1. Let us consider basic arithmetic on the set of integers $\mathbb{Z} = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$. In basic arithmetic, $\mathbb{Z}$ is endowed with two basic operations (let's ignore any other operations): addition (represented with "+") and multiplication (represented with "×"). Write first-order statements that represent the basic properties of arithmetic (this process is called "axiomatization"):

   (a) Both addition and multiplication are commutative.

   (b) $\mathbb{Z}$ is closed under addition and multiplication.

   (c) Multiplication distributes over addition.

   (d) Multiplication and addition are associative.

   (e) Multiplication and addition have the identity property.

2. Consider the following two sentences in first-order logic:
   $\alpha = \forall x(P(x) \vee Q(x))$, and
   $\beta = \forall x P(x) \vee \forall x Q(x)$.
   Does $\alpha \models \beta$? Justify your answer either with a proof, or a specific model where α is true and β is not.

3. Consider the following assumptions.

   (a) Han Solo owns the Millennium Falcon.

(b) Princess Leia is unhappy.

(c) Princess Leia loves Han Solo.

(d) For all x, if x owns the Millennium Falcon or x is unhappy then x visits Obi-Wan Kenobi.

(e) For all x, if x visits Obi-Wan Kenobi then x is wise.

(f) For all x, if x is owns the Millennium Falcon and visits Obi-Wan Kenobi, then Obi-Wan Kenobi teaches x to use the lightsaber.

(g) For all x, if x is unhappy or x owns the Millennium Falcon, and if Obi-Wan Kenobi teaches x to use the lightsaber, then x joins the Rebel Alliance.

(h) For all x and y, if x is unhappy and x loves y then x declares love for y.

(i) For all x and y, if Obi-Wan Kenobi teaches x to use the lightsaber and y declares love for x and y is wise, then x has Chewbacca as a friend.

Using these assumptions,

(i) use backward chaining to establish that Han Solo has Chewbacca as a friend.

(ii) use forward chaining to establish that Han Solo has Chewbacca as a friend.

4. Consider the following assumptions:

   (a) If Zeus were able and willing to prevent evil, then he would so.

   (b) If Zeus were unable to prevent evil, then he would be impotent.

   (c) If he were unwilling to prevent evil, then he would be malevolent.

   (d) Zeus does not prevent evil.

   (e) If Zeus exists, he is neither impotent nor malevolent.

   Prove that Zeus does not exist using resolution.

5. For each of the following first-order logical statements, show that the statement is valid by showing that its negation leads to a contradiction.

   (a) $\forall x(P(x) \Rightarrow P(x))$.

   (b) $(\neg\exists x P(x)) \Rightarrow (\forall x \neg P(x))$.

   (c) $(\forall x(P(x) \vee Q(x))) \Rightarrow ((\forall x P(x)) \vee (\exists x Q(x)))$

Part 2: Programming problem

The following problem requires you to write a program in Python. As in the past, you may import standard Python modules for routines not related to AI, such as basic sorting algorithms and basic data structures like queues. You must write all of the rest of the code yourself. If you have any questions about this policy, please ask us. We recommend using the CS Linux machines (e.g. `burrow.soic.indiana.edu`). You may use another development platform (e.g. Windows), but make sure that your code works on the CS Linux machines before submission. Remember to include a detailed comments section at the top of your code

that describes: (1) a description of how you formulated the problem and how your solution works, (2) any problems you faced, assumptions you made, etc., and (3) a brief analysis of how well your program works and how it could be improved in the future.

In an effort to improve public safety, the U.S. Federal Communications Commission (FCC) decides to open up new wireless radio frequencies to be used by the government of each U.S. state for emergency communication purposes. Ideally each state would receive its own unique frequency to avoid potential interference, but there are two problems with this. First, there isn't enough free bandwidth for 50 new frequencies; in fact, there's only room for 4. Second, some states (fortunately, relatively few) have legacy communication equipment that only works on one particular frequency, and unfortunately some of these frequencies are the same across states. Fortunately, as an enterprising young consultant, you realize that since radio waves have limited range, it's sufficient to ensure that nearby states do not share the same frequencies.

Write a Python program that assigns a frequency A, B, C, and D to each state, subject to the constraints that (1) no two *adjacent* states share the same frequency, and (2) the states that have legacy equipment that supports only one frequency are assigned to that frequency. Your program should be run like this:

`python radio.py` *legacy_constraints_file*

where *legacy_constraints_file* is an input to your program and has the legacy constraints listed in a format like this:

```
Indiana A
New_York B
Washington A
```

The output from your program should be a file called `results.txt` which lists all fifty states and a frequency, in a format like:

```
Alabama C
Alaska A
Arkansas D
```

and so on. Your code can also display results or debugging information to the screen, but the final line of output should be:

`Number of backtracks:` x

where x is the number of times your algorithm backtracked. To help get you started, we've included a file called `adjacent-states` in your repo that, for each state, lists the states that are adjacent to it. We've also included a few sample legacy constraint files for you try.

Grading. Most of the grade for this part will be based on coming up with a correction solution, but a small part will be based on a competition on how *fast* your code is. We will measure "wall-clock" time, i.e. the number of seconds between when your code is executed and when it finishes. We will run these tests on burrow or a similarly configured machine.

What to turn in

Turn in the files required above by simply putting the finished version (of the code with comments and PDF file for the first question) on GitHub (remember to `add`, `commit`, `push`) — we'll grade whatever version you've put there as of 11:59PM on the due date. To make sure that the latest version of your work has been accepted by GitHub, you can log into the github.iu.edu website and browse the code online.