Homework #2 - Logical Agents

COEN 266 - Artificial Intelligence

Spring 2017

Overview

For this homework assignment, you will be implementing a propositional logic resolution-based inference engine. There are 3 distinct parts, and you are free to complete as much as you are comfortable with, but please do not start a later portion without first completing all earlier ones.

Part 1: Two-Statement Resolution (70 points)

For this part of the assignment, you will be implementing a simple resolution function that resolves two sentences that are already in CNF.

Syntax

We'll use lists to represent sentences in propositional logic form. Each list will be represented as prefix operators followed by operands. Operators and operands will both be represented by Python strings.

Boolean Operator	Python Representation	# of operands
¬	"not"	1
∧	"and"	≥ 2
∨	"or"	≥ 2
⇒	"implies"	2
⇔	"biconditional"	2

Here are some examples of boolean expressions and their Python representation:

IsRaining	`"IsRaining"`
Studying ⇒ GoodGrades	`["implies", "Studying", "GoodGrades"]`
(a ∨ b) ⇔ ¬ (c ∧ ¬d)	`["biconditional", ["or", "a", "b"], ["not", ["and", "c", ["not", "d"]]]]`

API

Implement a single function:

`resolve` Accepts two arguments, both sentences in CNF. Returns:

- The resolution if the sentences can resolve against each other
- An empty list if the sentences resolve to a contradiction
- False if the two sentences cannot resolve

Demo

```
>>> print resolve(["or", "a", "b", "c"], ["not", "b"])
['or', 'a', 'c']
>>> print resolve(["or", "a", "b", "c"], ["or", "b", ["not", "c"]])
['or', 'a', 'b']
>>> print resolve(["or", ["not", "raining"], "wet ground"], "raining")
wet ground
>>> print resolve(["or", "a", "b"], "c")
False
>>> print resolve("a", ["not", "a"])
[]
```

Part 2: Resolution Inference Engine (20 points)

For this part of the assignment, you'll implement a complete resolution inference engine. It should accept CNF sentences in the same form as part 1 of this assignment, and answer queries by following the resolution algorithm. As a reminder, that means introducing the negation of the query and resolving until determining that the KB with inclusion of the query is either satisfiable (no new resolutions are possible) or contradictory.

API

Implement the following functions:

`TELL` accepts one argument, a sentence in CNF. Adds that sentence to the KB.

`ASK` accepts one argument, a single proposition or negated proposition. Returns True (indicating that the query must be true), or False (indicating that it cannot determine if the query is true).

`CLEAR` Removes all sentences from the KB.

Demo

```
>>> TELL(["or", ["not", "a"], "b"])
>>> TELL(["or", ["not", "b"], "c"])
>>> TELL("a")
>>> ASK("c")
True
>>> ASK("d")
False
```

Part 3: CNF Conversion (20 points)

For this part of the assignment, you'll augment your APIs to allow sentences using the full set of propositional logic operators, so you will have to implement the routines to convert expressions to CNF. After this is implemented, both the TELL and ASK functions should allow the specification of arbitrary expressions in the full propositional logic syntax. The values returned from ASK should not change, however.

Demo

```
>>> TELL(["implies", "a", "b"])
>>> TELL(["implies", "b", "c"])
>>> TELL("a")
>>> ASK("c")
True
>>> ASK("d")
False
>>> ASK(["implies", "c", "a"])
True
>>> ASK(["implies", "d", "a"])
True
```

Testing

In order to facilitate automated testing, all tests that you provide should use the naming pattern:

<testname>`-part`<assignment-part>`-`<username>`.py`

For example for a test that I wrote to verify portion 2 of this assignment, I might use the filename:

```
test1-part2-jconner.py
```

For each test, also provide an expected results file, which is the output of running that test. It should have the same name as the testcase, but with a `.out` file extension.

In order for the test harness to work properly, implement your routines in a file (or included from a file) named `hw2.py`.

Note

All code must run on the Engineering Design Center Linux machines (`linux.dc.engr.scu.edu`) without additional packages installed. Note that these machines are running a fairly ancient (2.6.6.) version of Python, so if you do any development outside of the DC computers, make sure to test early and test often!

Grading

You will be graded on correctness and coding style. The assignment will be graded out of 100 points, so anything above that is equivalent to extra credit.

Additional Credit

(5 points) The first person to identify each significant shortcoming in the assignment itself (this document) will receive a 5 point bonus.

(2 points) Each test case you submit that identifies a problem in another student's implementation will receive 2 points, up to a maximum of 10 points per student.

Submission

The assignment is due Monday, May 22 at 8:00am. All submissions should be made on Camino.