lab2sol

EECS 70

September 11, 2014

1 Virtual Lab 2 Solution: Logic and Quantifiers

EECS 70: Discrete Mathematics and Probability Theory, Fall 2014

Due Date: Monday, September 15th, 2014 at 12pm

Instructions:

- Complete this lab by filling in all of the required functions, marked with `"YOUR CODE HERE"`
- If you plan to use Python, make sure to go over **Tutorial 1A: Introduction to Python and IPython** before attempting the lab
- Most of the solution requires no more than 5 lines each
- Please do not hardcode the result or change any function without the `"YOUR CODE HERE"` mark
- Questions? Bring them to our Office Hour and/or ask on Piazza
- Good luck, and have fun!

1.1 Table of Contents

The number inside parentheses is the number of functions you are required to fill out for this lab. Always make sure to double check before you submit.

- Question 1: Truth Table Warm-up (3)
- Question 2: Quantifiers (2)
- Question 3: More Quantifiers (2)

Question 1: Truth Table Warm-upYou can create propositional statements in Python, as shown in the example below.

In [72]:
```
%%latex
$$x = a \vee \neg{b}$$

$$y = \neg{a} \wedge b$$
```

$$x = a \vee \neg b$$

$$y = \neg a \wedge b$$

In [73]:
```
def x(a, b):
    return a or not b
```

```
def y(a, b):
    return not a and b
```

In [74]:
```
def print_truth_table(var_list, *args):
    """
    Prints a truth table using the variables in var_list and len(args)
    propositional statements constructed from these variables

    Don't worry if this function is too complicated, you are only
    calling it and it basically prints out the truth table in a nice format
    """

    parts = list(var_list)
    parts.extend([s.__name__ for s in args])

    # Prints header of truth table
    header = "%3s" * len(var_list) + "  |" + "%3s" * len(args)
    body = "%3d" * len(var_list) + "  |" + "%3d" * len(args)
    print header % tuple(parts)

    # Prints body of truth table
    import itertools
    products = itertools.product([0, 1], repeat=len(var_list))
    for p in products:
        print body % (p + tuple([f(*p) for f in args]))
```

In [75]:
```
print_truth_table(["a", "b"], x, y)
```

```
  a  b  |  x  y
  0  0  |  1  0
  0  1  |  0  1
  1  0  |  1  0
  1  1  |  1  0
```

It's your turn. Fill in the functions f, g, and h as defined below. Handcraft a truth table on paper to make sure that the result is consistent.

In [76]:
```
%%latex
$$f = (a \wedge \neg{b}) \vee c$$

$$g = (\neg{a} \vee b) \wedge \neg{c}$$

$$h = (a \vee c) \wedge (\neg{b} \vee c)$$
```

$$f = (a \wedge \neg b) \vee c$$

$$g = (\neg a \vee b) \wedge \neg c$$

$$h = (a \vee c) \wedge (\neg b \vee c)$$

In [77]:
```
def f(a, b, c):
    """YOUR CODE HERE"""

    return (a and (not b)) or c

def g(a, b, c):
    """YOUR CODE HERE"""

    return ((not a) or b) and (not c)

def h(a, b, c):
    """YOUR CODE HERE"""
```

In [78]:
```
    return (a or c) and (not b or c)
print_truth_table(["a", "b", "c"], f, g, h)
```

```
   a  b  c  |  f  g  h
   0  0  0  |  0  1  0
   0  0  1  |  1  0  1
   0  1  0  |  0  1  0
   0  1  1  |  1  0  1
   1  0  0  |  1  0  1
   1  0  1  |  1  0  1
   1  1  0  |  0  1  0
   1  1  1  |  1  0  1
```

Comment on the relationship between the functions g, h, and f:

- g is the negation of f. Try to negate f and apply De Morgan's Laws and see what you get.
- h is logically equivalent to f by applying the distributive property to f.
- g is also the negation of h, since f and h are logically equivalent.

Question 2: QuantifiersImplement your own `forall` and `exists` functions, each takes in a list and a predicate and returns whether all/there is one element in the list that satisfies the predicate.

You can assume the predicate will take one parameter and return True/False based on whether the parameter satisfies the predicate or not.

DO NOT USE PYTHON'S BUILT-IN `any()` AND `all()` FUNCTIONS!

Hint: There is no need to maintain a count. Lookup short-circuit evaluation.

In [79]:
```
def forall(lst, pred):
    """YOUR CODE HERE"""

    for elem in lst:
        if not pred(elem):
            return False
    return True
```

In [80]:
```
def exists(lst, pred):
    """YOUR CODE HERE"""

    for elem in lst:
        if pred(elem):
            return True
    return False
```

Test your implementation below, and compare them with Python's built-in `any()` and `all()` functions

In [81]:
```
def Q2():
    lst = [-1, 1, 3, 5, 7]

    # forall
    ans1 = forall(lst, lambda x: x % 2 == 1)
    ans2 = forall(lst, lambda x: x > 0)
    if ans1 and not ans2:
        print "Great job with the implementation of forall!"
    else:
        print "Double check your forall implementation!"

    # exists
    ans3 = exists(lst, lambda x: x < 0)
    ans4 = exists(lst, lambda x: x % 2 == 0)

    if ans3 and not ans4:
```

```
        print "Great job with the implementation of exists!"
    else:
        print "Double check your exists implementation!"

    return ans1, ans2, ans3, ans4
```

In [82]:
```
m2 = Q2()
```

```
Great job with the implementation of forall!
Great job with the implementation of exists!
```

In [83]:
```
def Q2_builtin():
    lst = [-1, 1, 3, 5, 7]
    ans1 = all((lambda x: x % 2 == 1)(x) for x in lst)
    ans2 = all((lambda x: x > 0)(x) for x in lst)
    ans3 = any((lambda x: x < 0)(x) for x in lst)
    ans4 = any((lambda x: x % 2 == 0)(x) for x in lst)
    return ans1, ans2, ans3, ans4
```

In [84]:
```
m2 == Q2_builtin()  # this should print True
```

```
True
```

Out [84]:

Hey, you just implemented two built-in Python functions in the second lab. Good job! ## Question 3: More Quantifier-sImplement `forall2` using `exists`, and `exists2` using `forall`. Make sure to review the relationship between the two quantifiers before attempting this question.

Hint: Since `pred` is a predicate function that returns either True or False, you can't simply negate it by writing `not pred`. You need to make your own (anonymous) function by wrapping `pred`. Here's an example of an anonymous function, which you can read more about here

In [85]:
```
def is_positive(x):
    return x > 0

def my_func():
    return lambda x: not is_positive(x)
```

In [86]:
```
is_positive(10)
```

```
True
```

Out [86]:

In [87]:
```
is_not_positive = my_func()
```

In [88]:
```
is_not_positive(10)
```

```
False
```

Out [88]:

In [89]:
```
is_not_positive(-5)
```

```
True
```

Out [89]:

Now it's your turn. Fill in the implementation for `forall2` and `exists2` below.

In [90]:
```
def forall2(lst, pred):
    """YOUR CODE HERE"""

    return not exists(lst, lambda x: not pred(x))
```

In [91]:
```
def exists2(lst, pred):
    """YOUR CODE HERE"""

    return not forall(lst, lambda x: not pred(x))
```

In [92]:
```
def Q3():
    # We're going to use the same example as before again!
    lst = [-1, 1, 3, 5, 7]

    # forall2
    ans1 = forall2(lst, lambda x: x % 2 == 1)
    ans2 = forall2(lst, lambda x: x > 0)
```

```
    if ans1 and not ans2:
        print "Great job with the implementation of forall2!"
    else:
        print "Double check your forall2 implementation!"

    # exists2
    ans3 = exists2(lst, lambda x: x < 0)
    ans4 = exists2(lst, lambda x: x % 2 == 0)

    if ans3 and not ans4:
        print "Great job with the implementation of exists2!"
    else:
        print "Double check your exists2 implementation!"

    return ans1, ans2, ans3, ans4
```

In [93]:
```
Q3() == Q2_builtin()  # this should again print True
```

```
Great job with the implementation of forall2!
Great job with the implementation of exists2!
True
```

Out [93]:

Congratulations! You are done with Virtual Lab 2.

Don't forget to convert this notebook to a pdf document, merge it with your written homework, and submit both the pdf and the code (as a zip file) on glookup.

Reminder: late submission are NOT accepted. If you have any technical difficulty, resolve it early on or use the provided VM.