CS 2051: Honors Discrete Mathematics
Spring 2023 Homework 2 Supplement

Nithya Jayakumar
Written Assignment Lead

Sarthak Mohanty
Programming Assignment Lead

Written Assignment (10pts)

Show that $\forall a P(a) \vee \forall a Q(a)$ and $\forall a \forall b (P(a) \vee Q(b))$, where all the quantifiers have the same nonempty domain, are logically equivalent.

Submission Instructions: Submit a pdf or image of your answer to Gradescope.

Programming Assignment: Logic Playground (15pts)

Your first "official" programming assignment is to reason about a few functions that prove some statements about propositions. You won't need any advanced knowledge of Python, but you should be familiar with sets, lists, and string parsing.

We'll briefly cover some of the methods we've wrote for you, and then move on to explaining the functions you have to write.

- `extract_variables(proposition)`: This method receives as input a proposition as a string, and outputs the variables that participate in it as a sorted list. (You already did this on the Homework 0 Supplement.)
- `evaluate(proposition, model)`: You already know that in Python you can easily evaluate boolean expressions i.e. propositions[1]:

```
>>> p = True
>>> q = False
>>> p and ((not p) or q)
False
```

  But, how can you evaluate arbitrary expressions without even knowing the names of the variables ahead of time? This question itself could be an entire supplemental, but luckily Python makes the job easy with the `eval()` function. We can pass in a model[2] into this method to evaluate a propositional function:

```
>>> eval(p and ((not p) or q), {'p': True, 'q': False})
```

  We are also providing you with some black magic to enrich Python's Boolean operators (and, or, not) with two additional operators: `|iff|` and `|implies|`. These correspond to the bi-conditional ($\Leftrightarrow$) and implication ($\Rightarrow$) operators, respectively. This method will allow you to more easily evaluate complex logical expressions, like so:

```
>>> evaluate('p and (p |implies| q)', {'p': True, 'q': False})
```

[1] In class we introduced propositions and propositional functions as two different entities. Allow me the luxury of abusing notation for this assignment, and using the two interchangeably.

[2] A **model** is a mapping over a set of variables S assigning a truth value to every variable in S.

- `truth_table(proposition)`: This method is your "ace in the hole", at least with respect to this assignment. Running it on a valid proposition looks something like this:

  ```
  >>> truth_table('(p |implies| True) and q')
  [({'p': True, 'q': True}, True),
   ({'p': True, 'q': False}, False),
   ({'p': False, 'q': True}, True),
   ({'p': False, 'q': False}, False)]
  ```

  More formally, if the expression has n variables, the truth table returns a list with 2^n tuples, each tuple containing the model and the corresponding evaluation for the given proposition.

Now it's your turn to pick up the keyboard. We've tasked you to write six (well, really four) additional functions that determine the following properties of a proposition:

- `count_satisfying(proposition)`: receives a Boolean expression in exactly the same format as `truth_table` and returns the number of assignments of the variables for which the result is True. So for example, `count_satisfying('p and q')` should return 1.
- `are_equivalent(proposition1, propositions2)`: receives two Boolean expressions as input and returns a Boolean value that indicates if the two expressions are logically equivalent. For example, `are_equivalent('(not p) or q', 'p |implies| q')` should return `True`. Note that if two expressions do not have the same variables, they are not considered equivalent. For example, `"(not p) or q"` is not logically equivalent to `"x |implies| y"`'.
- `is_tautology(proposition), is_contradiction(proposition), is_contingency(proposition)`: receives a Boolean expression in exactly the same format as `truth_table` and returns a Boolean value that indicates whether the expression is a tautology, contraction, or contingency, respectively. For example, `is_tautology('p and q')` should return `False`, but `is_contingency('p and q')` should return `True`.
- `model_fitting(truth_table)`: If the `truth_table` method asks you to generate a truth table from a given proposition, this method asks you to work backwards: given a truth table, return a corresponding proposition. There will always be one such proposition. ,t *Hint:* You should only need the $\wedge$, $\vee$, and $\neg$ operators.

Submission Instructions: Download the template file and implement the six functions specified above; you can add functions of your own as needed (but no extra imports!). Submit the file logic.py to Gradescope and ensure all the test cases are passed.

Note: The autograder may not reflect your final grade on the assignment. We reserve the right to run additional tests during grading.