Programming (G54PRG)
4th exercise

Thorsten Altenkirch and Isaac Triguero

November 11, 2019

This exercise is about object oriented programming in Python. Your task is to use objects to implement a representation of Boolean expressions which can be printed and evaluated. On top of this, you should implement a method to produce truth tables and a tautology checker which determines whether the expression is *true* for every assignment of truth values.

1. In particular you are supposed to implement the following classes:

   `Expr` superclass for all boolean expressions,

   `Not` represents logical negation,

   `And` represents logical and,

   `Or` represents logical or,

   `Eq` represents logical equivalence,

   `Var` represents logical variables

   with constructors, so that, we can define the following expression trees:

   ```
   e1 = Or(Var("x"),Not(Var("x")))
   e2 = Eq(Var("x"),Not(Not(Var("x"))))
   e3 = Eq(Not(And(Var("x"),Var("y"))),Or(Not(Var("x")),Not(Var("y"))))
   e4 = Eq(Not(And(Var("x"),Var("y"))),And(Not(Var("x")),Not(Var("y"))))
   ```

2. Each class should implement an `__str__` method, so that, the above expressions print as follows:

   ```
   x|!x
   x==!!x
   !(x&y)==!x|!y
   !(x&y)==!x&!y
   ```

   It is ok but not perfect if you implement a version that prints too many brackets (that is better than printing not enough).

   To minimise the number of brackets you should take into account that:

(a) `Not` (`!`) binds stronger than `And` (`&`), e.g.

```
>>> print(And(Not(Var("p")),Var("q")))
!p&q
>>> print(Not(And(Var("p"),Var("q"))))
!(p&q)
```

(b) `And` (`&`) binds stronger than `Or` (`|`),e.g.

```
>>> print(Or(And(Var("p"),Var("q")),Var("r")))
p&q|r
>>> print(And(Var("p"),Or(Var("q"),Var("r"))))
p&(q|r)
```

(c) `Or` (`|`) binds stronger than `Eq` (`==`),e.g.

```
>>> print(Eq(Or(Var("p"),Var("q")),Var("r")))
p|q==r
>>> print(Or(Var("p"),Eq(Var("q"),Var("r"))))
p|(q==r)
```

This order is transitive, e.g. `!` binds stronger than `|` and `==`, and so on.

All binary operations (including `==`) are associative, hence there is no need to use brackets when only one kind of operation is used. E.g.

```
>>> print (And(Var("x"),And(Var("y"),Var("z"))))
x&y&z
>>> print (And(And(Var("x"),Var("y")),Var("z")))
x&y&z
```

3. Implement a method `make_tt` which returns the truthtable as a string. That is for example `print(e4.make_tt())` should produce the following output:

```
y       | x     | !(x&y)==!x&!y
True    | True  | True
False   | True  | False
True    | False | False
False   | False | True
```

The order in which you produce the lines doesn't matter.

The method should work for any collections of variables, not just for two called `x` and `y`.

4. A proposition is called a tautology if it is always *true*. That is, the truthtable contains only `True|` in the last column. Implement a method `isTauto` which determines whether the proposition is a tautology. E.g.

```
>>> e1.isTauto()
True
>>> e4.isTauto()
False
```

As above, this method should work correctly for propositions with any collection of variables.

Test your code with the test in `test04.py` which can be downloaded from Moodle. However, it should also work for other examples.

To get full marks your program should work, be well documented, not be unnecessarily complicated and you should use inheritance to avoid code duplication.

Important notes:

- Use only the operations that have been introduced in the lectures.
- Make sure you note down the name of the demonstrator.
- We will not give you the marks immediately.
- **Submission deadline: 26th of November**.
- **Demo deadline: 2nd of December**.