CSE 512 LABORATORY – Week 8, Winter 2016

Prof. Kerstin Voigt

In this lab we will attempt to implement the the CNF-conversion algorithm for propositional logic. The conversion proceeds in three stages:

1. Eliminate all implications:

   ```
   A => B   --->  notA or B
   ```

2. Move negations in :

   ```
   not(A and B)  --->  notA or not B

   not(A or B)    --->  not A and notB
   ```

3. Distribute 'and' over 'or':

   ```
   (A and B)  or C  --->  (A or C) and  (B or C)
   ```

For easier coding, we adopt a list representation for our propositional statements. A literal will be a a string of the from `P`, or `notP`. Examples of conjunctions, disjunctions, implications and negated propositions are:

```
['A', 'and', 'B']        ['A', 'or', 'B']       ['A', '=>', 'B']       ['not'
```

Such lists can be nested to arbitrary depth in order to repesent more complex propositional statements. E.g., imagine that `'A'` and/or `'B'` are propositions in list-form themselves.

You may start setting up your module with the following functions:

```
def is_literal (x):
    return type(x) == str

def is_neglit(x):
    return is_literal(x) and x[:3] == 'not'

def is_and(x):
    return type(x) == list and len(x) == 3 and x[1] == 'and'
```

```
def is_or(x):
    return type(x) == list and len(x) == 3 and x[1] == 'or'

def is_imp(x):
    return type(x) == list and len(x) == 3 and x[1] == '=>'

def is_negex(x):
    return type(x) == list and len(x) == 2 and x[0] == 'not'
```

Exercise 1: Implement a function `imp_elim` which eliminates all implications from the propositional expression.

```
def imp_elim(x):
# case 1: x is a literal

      # case 2: x is a disjunction (that may contain implications)

      # case 3: x is a conjunction (that may contain implications)

      # case 4: x is a negated statement (that may contain implications)

      # case5: x is an implication (that may  contain other impications)
```

Exercise 2: Implement a function `neg_in` which moves negations into the expressions so that the only negated statements will be literals.

```
def negs_in(x):
# case 1: x is a literal

      # case 2: x is a disjunction (that may contain negations)

      # case 3: x is a conjunction (that may contain negations)

      # case 4: x is a negated statement (that may contain other negations)
```

If there is time left, also do this last exercise.

Exercise 3: Implement a function `distrib_andor` which will distribute and-expressions over or-operators in order to produce an equivalent conjunction of disjunctions, which is the ultimate objective of the conversion.

```
def distrib_andor(x):
# case 1: x is an expression without 'and'

      # case 2: x is a conjunction (that may contain and-or's)

      # case 3: x is a disjunction (that may contain or produce and-or's)
```

Hand in: Submit evidence of your best effort (hardcopy of code and typescript) on **Thursday, March 10,** at the time of the lab.