CEN 598 Programming Assignment 2

In this assignment, you are to build a data structure and associated functions in python that represents a **Set of Subsets** (SOS).

1. Given is a set of **elements** $U = \{x_1, x_2, \ldots, x_n\}$.
2. Each element x_i is available in two forms, denoted by x_i and x_i'.
3. The empty set, denoted $\phi = \{\}$, is a set with no elements. It is a subset of every set.
4. A set with an empty set $\{\phi\} = \{\{\}\}$ has one member, namely ϕ, and is not empty.
5. Here are a few examples of SOS:
   (a) $\{\{a, b, c'\}\}$ (a set with one subset)
   (b) $\{\{a, b, a', c\}, \{c, b, d\}\}$ (a set with 2 subsets)
   (c) $\{\{a, b\}, \{a', c'\}, \{\phi\}\}$ (a set with 3 subsets)
6. Any subset that contains both forms of the same element must be eliminated. For example, $\{\{a, b, a', c\}, \{c, b, d\}\} = \{\{c, b, d\}\}$; $\{\{a, b, a', c\}\} = \{\} = \phi$.
7. If s_1 and s_2 are two subsets in a given SOS S, and if $s_1 \subseteq s_2$, then s_2 must be elminated.
8. An SOS is **reduced** if no element of any subset or no subset can be eliminated using the above two rules.
9. Two subsets s_1 and s_2 of an SOS are said to have a **consensus** if
   (a) there is exactly one element x (consensus element) that appears in one form in one of the subsets and the other form in the other subset. For example, $s_1 = \{a, b, c'\}$ and $s_2 = \{a, b, c, d\}$ have a consensus because only c occurs in one form in s_1 and and the other form in s_2. On the other hand, $s_1 = \{a, b, c'\}$ and $s_2 = \{a, b, d\}$ do not have a consensus; $s_1 = \{a, b, c'\}$ and $s_2 = \{a, b', c, d\}$ do not have an consensus.
   (b) if two subsets s_1 and s_2 have a consensus, then their consensus is the union of all the elements in s_1 and s_2 except for the consensus element. For example, the consensus of $s_1 = \{a, b, c'\}$ and $s_2 = \{a, b, c, d\}$ is $\{a, b, d\}$. Note that the consensus is contained in s_2. If they do not have a consensus, then the consensus is ϕ. We denote the consensus of s_1 and s_2 by $\#(s1, s2)$. (# is pronounced as sharp).
   (c) An SOS is **complete** if it is reduced and if any two subsets of the SOS have a consensus, then the consensus is included in the SOS.

Write a program in Python, that takes a number n denoting the set $U = \{x_1, x_2, \ldots, x_n\}$, and an SOS S over the set U, and returns a complete form of S.

1. your program should include, at a minimum, the following functions:

(a) *reduceS*(aSubset): reduces a subset, i.e., eliminates it if it contains both forms of an element

(b) *reduceSOS*(aSoS): returns an SOS that is a reduced version of the given aSOS.

(c) *completeSOS*(sSOS) which returns an SOS that is complete.

2. it is highly recommended that you create an object model of a subset and an SOS and simple primitive methods encapsulated in class definitions. Then write the above three functions.

3. Your code should be **elegant**, thoroughly documented (learn the principals of proper documentation from python tutorials) and should be written in a highly structured, top down manner. Construct class definitions, write small, simple functions and then write the main functions using the class methods.

4. Your code should also include a test function called *testdebug()* which sets up inputs and exercises the three functions described above. The last line of your python program should have

```
if __name__ == '__main__':
    testdebug()
```

   so that when your python program is simply executed, the function *testdebug* is executed. Otherwise it enters the interactive mode.

5. To simplify the input, you can use the function *testdebug* to create an input SOP. For instance, you can define a class describing a cube, which given a list of literals or their complements, will create an instance of a cube object. Then an SOP is a *set-of-cubes*. The SOP should be maintained as minimal wrt single cube containment. A true literal could simply be a string, whereas a complemented literal could be a special string that starts with a , e.g., *a* and *a*, or *foo* and *foo*.

6. For testing purposes, use *testdebug* to directly or manually create an SOP by calls to the class *cube*. For instance *testdebug* might be

```
def testdebug():
    c1 = cube(['a','b','c'])
    print('c1 = ',c1)  # would print a.b.c

    c2 = cube(['b','~e','d','c'])
    print('c2 = ',c2)  # would print as b.~e.d.c

    c3 = c1 + cube(['a','b','~d','c'])
    print('c3 = ',c3) # this would print abc
```

Note: *The above is just a suggestion for the input format. You would enter a arbitrary (properly defined) Boolean expression, and write a parser to create the internal representation and construct an SOP.*

This assignment will be graded based on elegance, documentation, simplicity, clarity and ofcourse, correctness of your code.