PE03: PE of 29/11/2019 (solutions)

Master in Informatics and Computing Engineering
Programming Fundamentals
Instance: 2019/2020

*An example of solutions for the 5 questions in this **Practical on computer evaluation**.*

1. Order Characters

Write a Python function called `reorder(l)` that receives a list `l` of tuples. Each tuple is of the form `(c, i)` where `c` is a character and `i` is the position where it should be placed in the string.

For example, `[('g', 3), ('d', 1), ('o', 2)]` should be rewritten as `"dog"`.

The indices are all different and cover the entire word.

Solution:

```
def reorder(l):
    r = [0] * len(l)
    for c, i in l:
        r[i-1] = c
    return ''.join(r)
```

2. Process commands

Write a Python function `process(commands)` that receives a list of `commands` to be processed over sets by the given order.

For example, `[{1, 3, 4}, '|', {2, 5}]` should be converted into `{1, 2, 3, 4, 5}`.

The following binary operations may be used:

1. "`|`": union
2. "`&`": intersection
3. "`x`": cartesian product
4. "`-`": except

Solution:

```
def process(commands):
    s = commands[0]
    for op, t in zip(commands[1::2], commands[2::2]):
        if op == '|': s = s | t
        if op == '&': s = s & t
        if op == '-': s = s - t
        if op == 'x':  # Cartesian product
            r = set()
            for i in s:
                for j in t:
                    r.add((i, j))
            s = r
    return s
```

3. Count digits

Write a Python function **`count_digits(n)`** that computes the frequency of each digit in the number **`n`**, with **`n`** > 0, and returns it in the form of a dictionary.

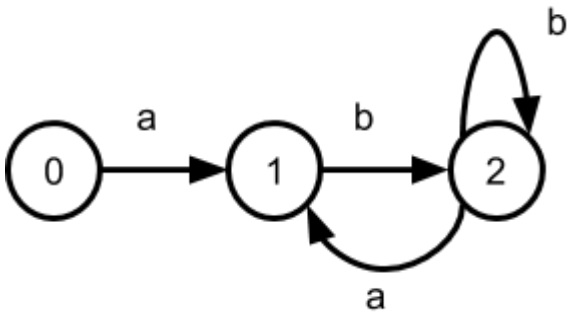
You cannot use cycles or global variables.

Solution:

```
def count_digits(n):
    if n == 0:
        return {}
    d = count_digits(n // 10)
    d[n % 10] = d.get(n % 10, 0) + 1
    return d
```

4. Finite-state machine

Write a Python function **`fsm(transitions, input)`** that receives a list of dictionaries in **`transitions`** and a string in **`input`**. The size of the list **`transitions`** corresponds to the number of states in the automata. For example, the automata in this picture:



is given by the list:

```
transitions = [
     {'a': 1},           # state 0
     {'b': 2},           # state 1
     {'a': 1, 'b': 2}    # state 2
]
```

With this automata, if **`input="a"`** then the function should return 1 (because it starts in state 0 and when it finds an "a" it transitions to 1). If **`input="ab"`** then it should return 2 (because it starts in state 0 then "a" forces a transition to 1, and then "b" forces a transition to state 2).

If there is an error (because the character is not recognized), then the function returns -1.

Solution:

```
def fsm(transitions, inputs):
    current_state = 0
    for c in inputs:
        if c not in transitions[current_state]:
            return -1
        current_state = transitions[current_state][c]
    return current_state
```

5. De Morgan

According to De Morgan's laws, an expression such as ¬(a ∧ ¬b) can be simplified to ¬a ∨ b.

Write a Python function **`simplify(expr)`** that given a logical expression **`expr`** simplifies it so that the negation occurs only at the variable level.

The logical expression **`expr`** is either a logical symbol (**`'a', ..., 'z'`**) or can be expressed recurrently as a tuple of the form:

1. `('¬', expr)`
2. `('∧', expr1, expr2)`
3. `('∨', expr1, expr2)`

For example, ¬(a ∧ ¬b) is represented by **`expr=('¬', ('∧', 'a', ('¬', 'b')))`**.

Solution:

```
def simplify(expr):
    if type(expr) != tuple:
        return expr
    if expr[0] == '¬':
        e = expr[1]
        if type(e) == tuple:
            if e[0] == '¬':
                return simplify(e[1])
            if e[0] == '∧':
                e1 = simplify(('¬', e[1]))
                e2 = simplify(('¬', e[2]))
                return ('∨', e1, e2)
            if e[0] == '∨':
                e1 = simplify(('¬', e[1]))
                e2 = simplify(('¬', e[2]))
                return ('∧', e1, e2)
        return expr
    return (expr[0], simplify(expr[1]), simplify(expr[2]))
```

The end.

FPRO, 2019/20