Problem Set #4

1 Theory Questions

Subjects: The pumping lemma for regular languages, Myhill-Nerode theorem.

1. For each of the following languages, prove that it is not a regular language using the pumping lemma:
   (a) $L_1 = \{a^n b^m : n, m \geq 0 \wedge m = 2n\}$ (over $\Sigma = \{a, b\}$)
   (b) Let $w \in \Sigma^*$ a string over an alphabet Σ. w^r notes the string that we get by reversing w. That is, if $w = w_1, ...w_n$ then $w^r = w_n, ...w_1$.
   $L_2 = \{w \in \{a, b\}^* : w^r = w\}$ (the palindromes language over $\Sigma = \{a, b\}$)
   (c) $L_3 = \{a^{n^2} : n \geq 0\}$ (over $\Sigma = \{a\}$)
   Note you may follow the proof we saw in recitation regarding the language $\{0^{2^n} : n \geq 0\}$ (over $\Sigma = \{0\}$).
2. For each of the following languages, prove that it is a regular language using Myhill-Nerode theorem:
   (a) $L_1 = \{w \in \{a, b\}^* : |w| \, is \, a \, multiply \, of \, 3\}$
   (b) $L_2 = \{w \in \{a, b\}^* : |a|_w = 1\}$
   (c) $L_3 = \{w \in \{a, b\}^* : w \, begins \, with \, "aa"\}$
3. For each of the following languages, prove that it is not a regular language using Myhill-Nerode theorem:
   (a) $L_1 = \{a^n b^{n+1} : n \geq 0\}$ (over $\Sigma = \{a, b\}$)
   (b) $L_2 = \{a^n b^m c^k : n, m, k \geq 0 \wedge (n = m \vee m = k)\}$ (over $\Sigma = \{a, b, c\}$)
   (c) $L_3 = \{a^{2^n} : n \geq 0\}$ (over $\Sigma = \{a\}$)
   Hint: look at the infinite set of strings $S = \{a^{2^n} : n \geq 0\}$. For every $x, y \in S$ such that $x \neq y$ define z to be the longer string between x and y.
4. Let L be a language over an alphabet Σ. We say that L *can be pumped* if there is a constant number P such that for every $w \in L$, if $|w| \geq P$ then there are strings $x, y, z \in \Sigma^*$ such that $w = xyz$, and the following conditions hold:
   - $|y| > 0$
   - $|xy| \leq P$
   - $\forall i \geq 0 : xy^i z \in L$

   For each of the following propositions, determine if it is true or false. Provide a short explanation (1-2 sentences):
   (a) If $index(L)$ is finite then L can be pumped.
   (b) If L can be pumped then $index(L)$ is finite.

2 Python programming

Subjects: Finite state machines (FSM).

Your task in the current problem set is to add an NFA implementation to the DFA implementation you have already created in problem set #3. We will point you to the objects within the **skeleton_for_fsm_module.py** file you are required to implement, using the specifications in this file.

Implement the following methods of the **NFA** class:

- **__init__**
- **decide**
- **get_dfa**

Implement the following method of the **DFA** class:

- **get_nfa**

Notes

1. Like in the DFA implementation, in the NFA constructor (the **__init__** method) the user is not required to explicitly define the alphabet of the NFA. Rather, this is inferred from the definition of the transition function.
2. Unlike the DFA implementation, in the NFA constructor the transition function passed by the user does not necessarily account for all possible combinations of current state and current character. In this case the NFA will reject an input string when reaching such a combination.
3. Like in the DFA implementation, the **decide** method should work also for input strings that contain characters that do not belong to the NFA's alphabet (and reject those strings).
4. The required complexity of the **decide** method is: run time - $O(n)$, space - $O(1)$ (when n is the length of the input string).

Running examples

```
>>> penultimate_is_b = NFA(
        ("initial", "saw_penultimate_b", "final"),
        {
            ("initial", "a"): ("initial",),
            ("initial", "b"): ("initial", "saw_penultimate_b"),
            ("saw_penultimate_b", "a"): ("final",),
            ("saw_penultimate_b", "b"): ("final",)
        },
        "initial",
        ("final",)
    )

>>> penultimate_is_b.decide("baabab")
False
>>> penultimate_is_b.decide("baabba")
True
>>> penultimate_is_b.decide("baaabc")
False                                          (according to note3 above)

>>> penultimate_is_b.get_dfa().decide("baabab")
False
>>> penultimate_is_b.get_dfa().decide("baabba")
True
>>> penultimate_is_b.get_dfa().decide("baaabc")
False

>>> contains_an_a.get_nfa().decide("bbbbbb")  (contains_an_a is defined in the
                                               running examples of ProblemSet3)
False
>>> contains_an_a.get_nfa().decide("bbbabb")
True
>>> contains_an_a.get_nfa().decide("bbbabc")
False
```