Problem Set #3

1 Theory Questions

Subjects: Closure properties of regular languages, NFA, $\varepsilon-$ transitions, DFA-NFA equivalency.

1. Let $\Sigma = \{a, b, c\}$. For each of the languages below, prove that it is regular by representing it as a result of applying a finite number of regular operations, on a finite number of finite languages (as we saw in the recitation).
   For example, the language $L = \{w \in \Sigma^* : w\, begins\, with\, 'a'\}$ is represented by $L = \{a\} \circ \Sigma^*$, and therefore is regular.

   If you want, you may use the following abbreviations in your representations:

   - Omitting the concatenation operation symbol '$\circ$'.
     For example, $\{a, b\} \circ \{c\} \circ \Sigma \circ \{b, c\}^* \circ \{abc\} = \{a, b\}\{c\}\Sigma\{b, c\}^*\{abc\}$.
   - Omitting the brackets symbols '()' that surround a union operand, if this operand does not contain any union operation symbol.
     For example, $(\{a, b\}^*\{c\}) \cup (\{a\}^*\Sigma) = \{a, b\}^*\{c\} \cup \{a\}^*\Sigma$.
   - Omitting the curly brackets symbols '{}' that surround a singleton language, if it is an operand of a concatenation.
     For example, $\mathbf{\{}a\mathbf{\}}\{b, c\}\{ab\}^*\Sigma^*\mathbf{\{}abc\mathbf{\}}\{a\}^*\mathbf{\{}b\mathbf{\}}\mathbf{\{}ab\mathbf{\}} = a\{b, c\}\{ab\}^*\Sigma^*abc\{a\}^*bab$.
     (*So* now for the language $L = \{w \in \Sigma^* : w\, begins\, with\, 'a'\}$ above we can write $L = \{a\} \circ \Sigma^* = a\Sigma^*$.
   - Omitting the curly brackets symbols '{}' that surround a 1-length singleton language, if it is an operand of a Kleene star operation.
     For example, $\mathbf{\{}a\mathbf{\}}^*bc\{a, b\}^*\{bc\}^*\mathbf{\{}b\mathbf{\}}^* = a^*bc\{a, b\}^*\{bc\}^*b^*$.

   (a) $L_1 = \{w \in \Sigma^* : |w|\, is\, even\}$
   (b) $L_2 = \{w \in \Sigma^* : w\, contains\, the\, string\, 'bbc' \vee the\, antepenultimate\, character\, of\, w\, is\, 'a'\}$
   (for a string $s = s_1...s_n$ with length $n \gneq 3$, the *antepenultimate character of* s is the character that appears in the third place from the end of s, namely the character s_{n-2}).
   (c) $L_3 = \{a^n b^m c^k : n, m, k \geq 0,\, n\, is\, even,\, m\, is\, odd,\, k\, is\, a\, multiply\, of\, 3\}$
   (d) $L_4 = \{w \in \Sigma^* : |a|_w = 1 \wedge |b|_w = 1\}$

2. Let L a language over an alphabet Σ . We define the language $DropChar(L)$ as follows:

$$DropChar(L) = \{xy \in \Sigma^* \mid \exists \sigma \in \Sigma : x\sigma y \in L\}$$

   (a) Prove that the regular languages are closed under the $DropChar$ operation. That is, prove that if L is a regular language then $DropChar(L)$ is a regular language.

   Hint: If N is a NFA with states set Q that decides L, define a NFA N' that decides $DropChar(L)$: using $Q \times \{0, 1\}$ as the states set of N' would enable to construct N' based on two duplicates of N. Define the initial state of N' to be the initial state of the first duplicate. Define the accepting states of N'. Define the transition function of N', and specifically use $\varepsilon-$ transitions to connect states of the first duplicate with states of the second duplicate.

   (b) Let $\Sigma = \{a, b, c\}$ and $L_{abc} = \{abc\}$ over Σ (a singleton language).

i. what is the language $DropChar(L_{abc})$?
ii. Use your general construction in (a) and construct a NFA that decides $DropChar(L_{abc})$.

3. Consider the following NFA with $\varepsilon-$ transitions $N = (Q,\Sigma,\delta,q_0,F)$ where:

- $Q = \{q_0,q_1,q_2,q_3,q_4\}$
- $\Sigma = \{a,b,c,d\}$
- q_0 is the initial state.
- $F = \{q_1,q_2,q_3,q_4\}$ is the set of the accepting states.
- the transition function $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ is defined by the table below.
(If for $(q,\sigma) \in Q \times \Sigma_\varepsilon$ no $\delta-$value appears in the table, $\delta(q,\sigma) = \phi$).

Q	Σ_ε	$P(Q)$
q_0	ε	$\{q_1,q_2\}$
q_1	a	$\{q_1\}$
q_1	ε	$\{q_3\}$
q_2	b	$\{q_2\}$
q_2	ε	$\{q_4\}$
q_3	c	$\{q_3\}$
q_4	d	$\{q_4\}$

(a) What is $L(N)$? (drawing N may be helpful, but you are not required to submit your drawing).
(b) For every $q \in Q$ compute $E(q)$ (the $\varepsilon-$closure of q). Provide a short explanation.
(c) Let $D = (Q',\Sigma,\delta',q_0',F')$ a DFA that is equivalent to N and is generated by the algorithm we saw in the recitation.
i. Compute q_0'. Provide a short explanation.
ii. For every $\sigma \in \Sigma$ compute $\delta'(q_0',\sigma)$. Provide a short explanation.

2 Python programming

Subjects: Finite state machines (FSM).

Your task in the current problem set (#3) and in the next problem set (#4) is to create a Python module that implements FSM - finite state machines (DFA and NFA). In the current problem set you would implement DFA, and in the next one you would implement NFA.

The file **skeleton_for_fsm_module.py** in the Piazza website contains a detailed and exact specifications of the module. You are not required to implement all the objects that appear in the skeleton. Rather, we will point you to the specific objects you are required to implement.

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

- **__init__**
- **decide**
- **get_complement**
- **get_union**
- **get_intersection**

Notes

1. The skeleton mentions sequences and mappings. A sequence is any type like **tuple** and **list** that supports iterations (with a **for** loop), slicing and similar operations, A mapping is a type like **dict** that supports arbitrary key look-ups using square brackets.

2. In the DFA constructor (the **__init__** method), as required in DFA definition, the transition function passed by the user accounts for all possible combinations of current stare and current character. In addition, the user is not required to explicitly define the alphabet of the DFA. Rather. this is inferred from the definition of the transition function.

3. The **decide** method should work also for input strings that contain characters that do not belong to the DFA's alphabet (and reject those strings).

4. In the **get_union** and **get_intersection** methods, assume that the arguments are DFAs with the same alphabet.

Running examples

```
>>> contains_an_a = DFA(
        ("initial", "saw_a"),
        {
            ("initial", "a"): "saw_a",
            ("initial", "b"): "initial",
            ("saw_a", "a"): "saw_a",
            ("saw_a", "b"): "saw_a"
        },
        "initial",
        ("saw_a",)
    )

>>> contains_an_a.decide("bbbbbb")
False
>>> contains_an_a.decide("bbbabb")
True
>>> contains_an_a.decide("bbbabc")
False                                   (according to note3 above)

>>> complement = contains_an_a.get_complement()

>>> complement.decide("bbbbbb")
True
>>> complement.decide("bbbabb")
False
>>> complement.decide("bbbabc")
False
```

```
>>> begins_with_b = DFA(
        ["initial", "begins_with_a", "begins_with_b"],
        {
            ("initial", "a"): "begins_with_a",
            ("initial", "b"): "begins_with_b",
            ("begins_with_a", "a"): "begins_with_a",
            ("begins_with_a", "b"): "begins_with_a",
            ("begins_with_b", "a"): "begins_with_b",
            ("begins_with_b", "b"): "begins_with_b"
        },
        "initial",
        ["begins_with_b"]
    )

>>> begins_with_b.decide("abbbb")
False
>>> begins_with_b.decide("baabab")
True
>>> begins_with_b.decide("bbcbbb")
False
```

```
>>> union = contains_an_a.get_union(begins_with_b)

>>> union.decide("bbb")
True
>>> union.decide("aaa")
True
>>> union.decide("")
False

>>> intersection = contains_an_a.get_intersection(begins_with_b)

>>> intersection.decide("bbb")
False
>>> intersection.decide("aaa")
False
>>> intersection.decide("bab")
True
```