FSM Lab 00000101

DFAs and NFAs

Your goal is to create the functions DFA and NFA that, given an input string and a corresponding FSM, return whether the input string is accepted or not.

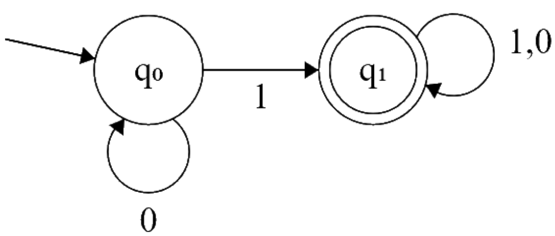
The function headers are:

```
(DFA input Sigma S s0 delta F)
(NFA input Sigma Q q0 Delta F)
```

As a reminder:

- Sigma is the alphabet of a FSM.
- S (or Q) is the list of states.
- S_0 (or q_0) is the start state
- Delta (or delta) is the list of transitions
- F is the list of final states.

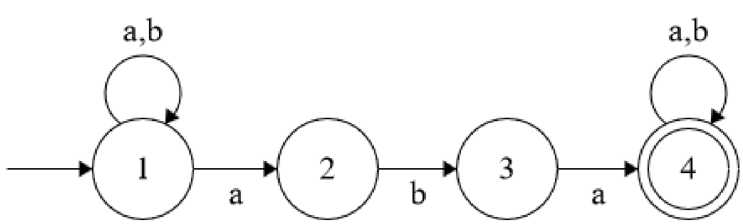
So, if we wanted to see whether "**0000**" would be accepted by this DFA…



…we would use:

```
(DFA "0000" '("1" "0") '(q0 q1) 'q0 '((q0 "0" q0) (q0 "1" q1) (q1 "0" q1) (q1 "1" q1)) '(q1))
```

And, if we wanted to see whether "**aaab**" would be accepted by this NFA…



…we would use:

```
(NFA "aaab" '("a" "b") '(t1 t2 t3 t4 NA) 't1
    '(
        (t1 "b" (t1)) (t1 "a" (t1 t2)) (t2 "b" (t3)) (t2 "a" (NA)) (t3 "b" (NA))
        (t3 "a" (t4)) (t4 "a" (t4)) (t4 "b" (t4)) (NA "a" (NA)) (NA "b" (NA))
     )
    '(t4))
```

Please note that our Delta function (for NFAs) maps a single input to multiple possible outputs. We will never map multiple states to multiple outputs within one δ, so you do not need to worry about any deltas such as **((t1 t2 t3) "a" (t2 t4))**. This should simplify your logic.

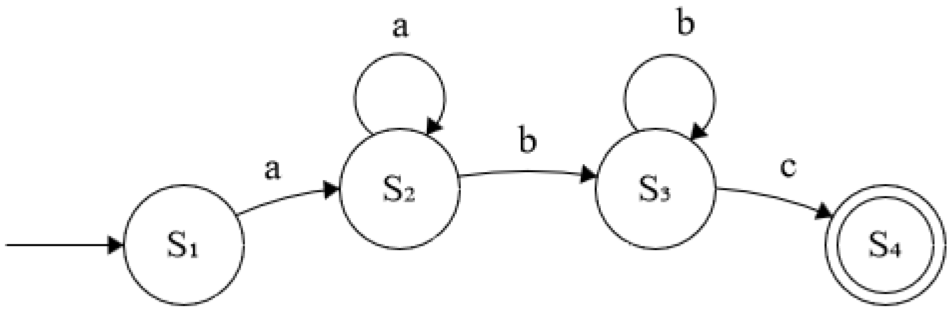
To streamline our coding, we will further stipulate that we will never use the same input from the same state in two different transitions. So,

```
[…   (t1 "a" (t1 t2)) (t1 "a" (t3 t4))   …]
```

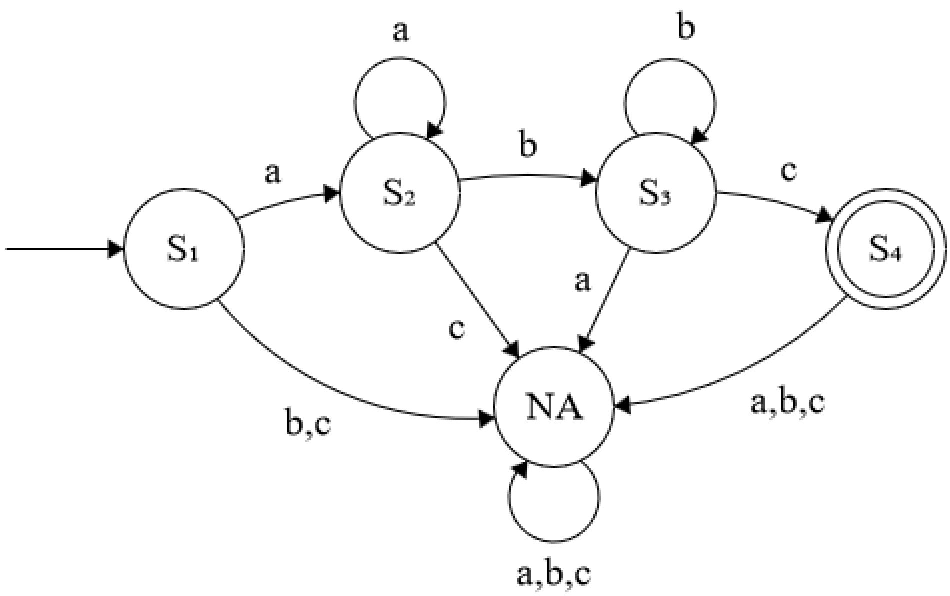
…would instead be written as…

```
[…   (t1 "a" (t1 t2 t3 t4))   …]
```

Furthermore, even though we can informally write out a machine like this:



… we will not be feeding machines with unwritten transitions into your function. **You can assume that every transition will be included**:



Many hints:

1) Since you are making a recursive function, *you can make use of* ***s0*** *to represent the current state instead of just the starting state, because execution starts at this state, and every recursive step starts in a new place*. If you don't understand this **s0** point, or why this is, **stop here.** Start to ask questions. Ponder. Consider.
2) Similarly, for NFAs, you can use **q0** to represent all of the possible current states. In order to do that, you will need to make a helper function that takes a list of states instead of a single state in place of **q0**. You can initially call this function with something like **(NFArunner input Sigma Q (cons q0 null) Delta F 0)**. This would be the only thing that **(NFA …)** actually does.

3) That last **`0`** is because there needs to be a maximum depth. Otherwise, I could give you an NFA with a transition **`(S2,"",S2)`**, and your machine would simply run forever. Define the max depth as a constant called **`max`** at the top of your code. You might want to use 50 as a reasonable starting value. If you reach max depth on some branch of your execution, and you're not on an accept state, that branch should not accept.
4) Speaking of that empty string, we will use the empty string to represent epsilon transitions in NFAs.
5) You can infer the alphabet from the transitions if you wish. You might not really need to make use of **`Sigma`** (though it will still be provided). Similarly, you might not need to pay much attention to **`S`**! They are part of the tuple because they are useful in proofs, not because they are used in the computation.
6) If you think about it, one of these machines could almost trivially be redefined as the other. You could, hypothetically, just make **`NFArunner`**, and have both **`DFA`** and **`NFA`** call that function with just a teeny bit of translation work. On the other hand, if you're not feeling terribly confident, it might be a good idea to actually write **`DFA`** as a way to get your legs back under you before you tackle the harder problem. It is your choice!
7) I made a helper function called **`finalstate?`** that was called only when I ran out of input string. I fed it my current **`s0`** (or all of my current **`q0`**s) to determine whether my current state was, in fact, an accept state.
8) Another useful helper function that I made was called **`nextstate`**. This function takes the current state, the list of delta transitions, and the current input, and returns the next state (or list of states – whatever is **`cddr`** of the relevant transition).
9) The only difference between an NFA and a DFA is that there are multiple possible states at once. Since our function ultimately returns a boolean, we don't care what NFA path we actually took to arrive at our answer. *If any possible path returns **`#t`**, then the overall answer should also be **`#t`**. What kind of boolean relationship makes sure that, if any input is true, that the output is also true?*