Project 2: NFA Simulator

CS 440: Programming Languages and Translators, Fall 2020

Due Mon Oct 23, 11:59 pm

11/11 pp.1,2, 11/17 p.2

How to Work; How to Submit

Same way as in project 1.

NFA Simulator

The goal is to extend your DFA simulator from project 1 to an NFA simulator. You'll have to augment the reading of state transitions to include ε-transitions, and you'll have to keep a collection of current states instead of just one state. Other than that, the basic structure is the same: read in the description of an NFA and an input string and run the NFA to see whether or not it accepts the input string. At each step, you print out the set of states you're in and the terminal symbol you saw. At the end, you print out your final set of states and say whether or not you accept the input. As before, the project can be in OCaml or python (your choice). It doesn't have to be in the same language as for project 1.

NFA Description Format

- The description format is the same as that for the DFA except that the transitions can be nondeterministic (more than one target for a state/symbol combination), and we can have ε-transitions.
- The ε-transitions will go between the regular transition cases and the line of input.
- So the input is
  - One line with the number of states....
  - ... same as in project 1
  - One line with the number of transition cases
    - [11/11] One line for each transition case, in %d,%c,%d format (old state, comma, symbol (possibly blank), comma, and new state.
  - [new to project 2]: One line with the number of ε-transitions
    - One line for each ε transition in %d,%d format (old state, new state).
  - One line with the input to the DFA [as in project 1].

NFA Processing

As in project 1, print out a trace of information about the NFA as you read it. If you notice an error, you're allowed to print an error message/raise an exception and halt. You're also allowed to wait

until execution of the NFA to realize there are errors in the transition (bad state number or bad symbol — it's now okay to have multiple entries for the same state/symbol pair).

When you execute the NFA, include all the current states the NFA can be in. Surround the list of states with parentheses or brackets or some sort of grouping system to make things clearer. Note if you're in state s and see symbol c, you have to find all new states t for s and c, plus you have to find all ε-transition states reachable from t (plus the ε-transition states reachable from them, and so on.)

Cases to Watch For

- Make sure your set of current states is really a set (no duplicate states), for a couple of reasons.
  - You can have more than one of your current states lead to the same target on a particular input. E.g., if you're in states {0, 1} and both of them have a transition on a to state 2, your new state should be {2}, not {2, 2}.
  - It's legal to have a cycle of ε-transitions (as in state 0 $\rightarrow_\varepsilon$ 1 and 1 $\rightarrow_\varepsilon$ 0, or even 0 $\rightarrow_\varepsilon$ 0. In that case, your start state is {0, 1}. (Not {0, 1, 0, 1, 0, 1, ... }.)
- Small Example
  - Say we have regular transitions
    - 0,a,2
    - 0,a,1
    - 0,a,0
    - 1,b,3
    - 2,b,4
  - And ε-transitions
    - 0,1
    - 0,3
    - 3,4
  - [11/17] Then our initial state is {0,1,3,4}. On a, we go to {0,1,2,3,4}, and from there, on b we go to {3, 4}.