Assignment 3 - Quantifiers, Asymptotics, and NFAs

CS 234

due Feb 14, 11:59pm

0 Introduction

This assignment comes in two parts. The first part is on paper. The second part is in code.

This assignment is to be completed individually, but feel free to collaborate according to the course's external collaboration policy (which can be found in the syllabus).

The deliverables consist of one `.pdf` file and one `.py` file. The deliverables should be submitted electronically to by the deadline. Put any attribution text in the `.pdf` file. You may also consider adding an experience report to the `.pdf` describing your experience with the assignment: how long did it take, how hard/fulfilling was it, etc.

Your `.pdf` file should be named like `FLast_cs234_aX.ext` where `F` is your first initial, `Last` is your last name, `X` is the assignment number, and `ext` is the appropriate file extension. For example, Laure Daviaud's `.pdf` file should be given the name `LDaviaud_cs234_assignment3.pdf`. (Laure Daviaud is an automaton researcher, and I met her at my first ever CS conference.)

Your `.py` file **must be named** `aX.py`, where `X` is the assignment number. You will be unable to get credit if you do not follow this naming convention.

1 Quantifiers and NFAs on Paper

Please complete the following exercises from the textbook in your `.pdf` submission. Clearly label your responses with the exercise number.

This part is worth **88 points** in total and has 22 questions.

By the way, if you don't want to draw your NFAs by hand, there are various automaton drawing utilities online, including `https://madebyevan.com/fsm/`.

- 3.47, 3.50, 3.52, 3.53
- 3.55, 3.56, 3.57, 3.58
- 3.65, 3.69, 3.71 (make sure to push the negation all the way into each proposition until you no longer need the $\neg$ symbol)
- 4.5, 4.7
- 4.8, 4.10, 4.13, 4.14, 4.16, 4.18, 4.19
- 4.20, 4.22

2 Coding with NFAs

Please complete the following tasks in your `.py` submission. This part is worth **12 points** in total. Make absolutely sure to implement the indicated function names, and do not import any Python libraries. Also, please to not have your file print when loaded as a module.

NFAs and ϵNFAs can of course be coded up and used to classify strings. They also can be converted to DFAs using the subset construction discussed in class. You will do exactly these activities in this task, with each function being worth **4 points** to implement.

You may want to refer to your code for Assignment 1 for working with functional relations, and you may want to refer to your code for Assignment 2 for working with DFAs. You may also want to recall details about working with `set`s in Python. In particular, `set`s cannot contain other `set`s, only `frozenset`s, and the order of set-like objects in Python is randomly chosen so cannot be relied upon.

1. Implement a function `acceptNFA` which takes two arguments:

   - a 5-tuple defining an NFA
     - Element 1 is the set of states Q. This is a Python set containing some other sort of Python data representing states.
     - Element 2 is the input alphabet Σ. This is a Python set containing strings of length 1.
     - Element 3 is the transition function. This is provided as a functional relation for a function of type $Q \times \Sigma \to \mathcal{P}(Q)$. *The only tuples in a functional relation are of length 2. The first element is a function argument (which may also be a tuple), and the second is the return for that argument.*
     - Element 4 is the start state. This is one of the elements in Q.
     - Element 5 is the set of final states. This is a subset of Q.
   - a string of only characters contained in the NFA's input alphabet Σ

   and returns a Boolean indicating whether the input NFA accepts the input string.

2. Implement a function `convertNFAtoDFA` which takes a single argument, the 5-tuple for an NFA, and returns a 5-tuple for a DFA with the same language.

3. Implement a function `acceptENFA` which takes two arguments:

   - a 5-tuple defining an ϵNFA
     - Element 1 is the set of states Q. This is a Python set containing some other sort of Python data representing states.
     - Element 2 is the input alphabet Σ. This is a Python set containing strings of length 1.

  - Element 3 is the transition function. This is provided as a functional relation for a function of type $Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$, where ϵ is represented by the Python empty string `""`. *The only tuples in a functional relation are of length 2. The first element is a function argument (which may also be a tuple), and the second is the return for that argument.*
  - Element 4 is the start state. This is one of the elements in Q.
  - Element 5 is the set of final states. This is a subset of Q.
- a string of only characters contained in the ϵNFA's input alphabet Σ

and returns a Boolean indicating whether the input ϵNFA accepts the input string.

You might consider using your code to test your answers to the written part!