ToC: Final Project [150 points]

Due: Wednesday, December 13th, by midnight

How to submit

Submission is via ICON. Please put your solution in a folder whose name is your Hawkid. If you worked with a partner, then both of you should include a file called `partner.txt` that lists the Hawkids for you and your partner. But only one partner should submit. You will both receive the same grade. Once you begin working on the project with a partner, you should continue with them through the end of the project. If you run into difficulties, email Prof. Stump.

How to get help

Please post questions to ICON Discussions, if possible (as opposed to emailing us directly), so that our answers can benefit others who likely have similar questions. You may also come to our Zoom office hours, listed on ICON under Pages - Office Hours.

Overview

You have two options for this project. They both involve implementing algorithms from the first two parts of the class. You are free to pick whichever one you wish, and you may use whichever programming language you prefer for your implementation. Naturally, the code should be original, written just by you (and your partner, if you have one). You may make use of standard library functions for basic data structures that you might need, but you should implement the algorithms yourself from scratch. Please refrain from considering similar implementations in the programming language you are using. It is ok, though, to study implementations in a different language, if you can find such.

The two options are listed in the next two sections. You should pick just **one** of them! The last section of these instructions describes requirements for testing and reporting.

Option 1: translating regular expressions [150 points]

The goal is to translate regular expressions into nondeterministic finite automata with epsilon-transitions, and then determinize those automata. Please follow these steps:

1. Define a datatype (in whatever language you are using) for representing regular expressions. [10 points]
2. Define a datatype for representing nondeterministic finite automata with epsilon-transitions (as this format is used for the first step of translating regular expressions). [15 points]

3. Write a printing function that can generate a GraphViz file for the transition diagram of an automaton as implemented in the previous step. I am providing a sample GraphViz file for an automaton, for inspiration. You can render such files using online tools like

   This is `dfa1.gv` (rendered in `dfa1.pdf`). [15 points]

4. Write a function to translate regular expressions into finite automata. [60 points]

5. Now write a function that can determinize a finite automaton. You can just take in an automaton of the type you defined, and generate a new one of those automata, where the transition relation is nondeterministic: no epsilon-transitions, and exactly one outgoing edge with each character of the input alphabet, for each state. The advantage of this approach is that you do not need to define a new datatype for DFAs. Just use the original datatype for NFAs, but use the subset construction to eliminate all nondeterminism. [50 points]

6. Write up your solution following the testing and reporting requirements at the end of these instructions.

Please note that there is no requirement to complete all parts above. It is ok to complete just some parts, and skip others (losing, of course, the possibility of earning the points for those parts that you skip). For example, if you have enough points already in the class for the grade you are shooting for, you might decide to skip the part about determinizing automata. I just am stating explicitly here that this is totally fine.

Option 2: algorithms on CFGs and PDAs [150 points]

Implement the following algorithms on CFGs and PDAs.

1. Define a datatype (in whatever language you are using) for representing context-free grammars. [10 points]

2. Define a datatype for representing push-down automata. [15 points]

3. Write a printing function that can generate a GraphViz file for the transition diagram of a PDA as implemented in the previous step. I am providing a sample GraphViz file for a PDA, for inspiration. This is `palindromes.gv` (rendered in `palindromes.pdf`). [15 points]

4. Write a function that can translate a CFG into a PDA, following the algorithm described in week 8 of the class (see the module for Oct. 10-12 on ICON). [60 points]

5. Write a function to transform a grammar into Chomsky normal form, following the algorithm discussed in week 8 of the class (see also Kozen Lecture 21). [50 points]

6. Write up your solution following the testing and reporting requirements at the end of these instructions.

As noted for option 1, it is fine to skip some parts. For example, you could still earn 100 points if you skipped the part about transforming grammars to Chomsky normal form.

Testing and reporting

To get full credit, you must describe your code, and test it.

For documentation, your submission must include a `README.txt` file in plain text (not Microsoft Office or other format) which has these sections:

- **Authors**: Your name and (if you have one) your partner's name
- **Tooling**: state which programming language you used to implement your solution, and any libraries that need to be installed to run it
- **Sources**: list and briefly describe the source files for your solution
- **Required functions**: list and briefly describe the functions you wrote to satisfy the various parts of the project (as listed above, in their respective sections)
- **Status**: Are all your functions working correctly, or are some incomplete or have bugs that you know of? Please give us your assessment, to help us as we review your code (so we don't waste time trying to puzzle out how something is working if it is incomplete or buggy).

Also, your submission must include a file called `TESTING.txt` that shows the results of your tests. Just break that file into sections where for each section, you state which test you ran and what the output was. If you generate some graphs in GraphViz format, then include those files as part of your submission, and reference them in this TESTING file.

The reason we are asking for this TESTING file is that we may not be able to try running your code ourselves, due to time limitations (especially with possibly needing to install new software). So we want you to show us what your code can do, by showing the results of calling the various functions, using the tests you wrote.