ProjectCSE105W24: Project

CSE105W24

Due February 22 at 5pm (no penalty late submission until 8am next day)

The CSE 105 project is designed for you to go deeper and extend your work on assignments and to explore an application of your choosing. The project is an individual assignment and has two tasks:

Task 1: Implementing the construction that converts NFA to DFA, and

Task 2: Illustrating the theorem that every regular language is decidable

What resources can you use? This project must be completed individually, without any help from other people, including the course staff (other than logistics support if you get stuck with screencast). You can use any of this quarter's CSE 105 offering (notes, readings, class videos, homework feedback). Tools for drawing state diagrams (like Flap.js and JFLAP) can be used to help draw the diagrams in the project too. These resources should be more than enough. If you are struggling to get started and want to look elsewhere online, you must acknowledge this by listing and citing any resources you consult (even if you do not explicitly quote them), including any large-language model style resources. Link directly to them and include the name of the author / video creator, any search strings or prompts you used, and the reason you consulted this reference.

The work you submit for the project needs to be your own. Again, you shouldn't need to look anywhere other than this quarter's material and doing so may result in definitions or notations that conflict with our norms in this class so think carefully before you go down this path.

If you get stuck on any part of the project, we encourage you to focus on communicating what you think the question might mean, including bringing an example from class or homework you think might be relevant, and include any submission any aspect where you're unsure. Clear communication about these theoretical ideas and their applications ia one of the main goals of the project.

Submitting the project You will submit a PDF plus a video file for the first task and a PDF for the second task. All file submissions will be in Gradescope.

Task 1: Implementing a construction Nondeterminism is a useful theoretical concept because it can make designs simpler and more modular. However, our actual devices are deterministic. In this part of the project, you'll choose a language that can be represented using nondeterminism in some interesting way, then illustrate the construction that converts NFA to DFA for this example.

Specifically:

1. Choose an alphabet Σ for this entire first task of the project.

2. Write a program in Java, Python, JavaScript, C++ , or another programming language of your choosing that takes as input a representation of an NFA over this alphabet and outputs a representation of a DFA over this alphabet that recognizes the same language. You get to choose the way NFA and DFA are represented, so long as it is general enough to represent any NFA and any DFA over this alphabet. For simplicity: you can restrict your attention to NFA **without spontaneous moves** (in other words, where the transition function has domain $Q \times \Sigma$ when Q is the set of states of the NFA). Informally, this means that the nondeterminism is coming from there being zero, one, or more arrows coming out of each state for each character in the alphabet and there are no arrows with ε labels.

   If you would like, you may use aids such as co-pilot or ChatGPT to help you write this program. However, you should test the code that is produced and be able to explain what it is doing. As a header in your code file, include a comment block describing any resources that were used to help generate your code.

   Whenever your program is run, it should display a representation of the input NFA and of the output DFA of the run.

3. To demonstrate your program, design a NFA over Σ with three states and with no spontaneous moves where the language of the NFA is neither $\emptyset$ nor Σ^* and draw its state diagram. Your NFA should use nondeterminism in some way. In other words, the state diagram you draw can't already be the state diagram of a DFA. Run your program with the NFA you just designed to output a representation of an equivalent DFA and demonstrate its design and the test case on video.

Checklist for submission

For this task, you will submit a PDF plus a video file.

The PDF should include:

- Clear specification of alphabet and state diagram of chosen three-state NFA.
- Documentation for program converting NFA to DFA: include a description of how NFA and DFA are represented in the program and give instructions for running it.

- Printout of code for program converting NFA to DFA.
- Screen shots of demonstration of running your program on your chosen NFA, including the representation of the output DFA.
- Solution is typed or clearly hand drawn with precise language and notation for all terms.

Presenting your reasoning and demonstrating it via screenshare are important skills that also show us a lot of your learning. Getting practice with this style of presentation is a good thing for you to learn in general and a rich way for us to assess your skills. To demonstrate your work, you will create a 3-5 minute screencast video with the following components:

- Start with your face and your student ID for a few seconds at the beginning, and introduce yourself audibly while on screen. You don't have to be on camera for the rest of the video, though it's fine if you are. We are looking for a brief confirmation that it's you creating the video and doing the work you submitted.
- Present the NFA you will be working with. Your video should include a clear and precise explanation of why the language of this NFA is not empty and also not the set of all strings over Σ.
- Show on the screen and explain the code for your program, including the software design choices you made (e.g. which data structures are you using, etc.) and any resources you used. The video should clearly describe which programming language was chosen for the implementation and gives the reasons why.
- Show on the screen and explain the representation of the NFA that you will input to the program.
- Demonstrate running your code on your example input. The video should include screencasts of running the code live. Explain why the output of your program is what you would expect, by connecting the output of the program to a DFA and discussing which strings are accepted / rejected by this DFA.
- Logistics: video needs to load correctly, be between 3 and 5 minutes, show your face and ID, and you introduce yourself audibly while on screen.

Note: Clarity and brevity are both important aspects of your video. In previous years, we've seen students speed up their videos to get below the 5 minute upper bound. This is ok so long as it doesn't compromise clarity.

Task 2: Illustrating a theorem In this part of the project, you'll choose a pattern in an application you care about, define it precisely, build a DFA that recognizes it, and then build the related Turing machine that proves that the language encoding this pattern is decidable.

First, pick **one** application for your example. Here are some ideas to get you started - but you can choose to go in a different direction.

- Data validation for input in text files (e.g. emails with specific domains, dates in specific formats, PIDs in a class list, etc.)
- Finding ASCII codes for punctuation in a binary file.
- The CDC recommended procedure for hand washing (Refer to the guidelines from the CDC here https://www.cdc.gov/handwashing/index.html in your explanation. You might find the first example in chapter 1 about automatic door controllers helpful when starting your design.)
- See more ideas here: https://theory-cs.github.io/files/practical-applications-of-theory-of-computation.pdf

Then:

1. In a paragraph or so, give the context for your chosen application and why you chose it.
2. Specify the alphabet for your example and write a precise (mathematical and/or English) description of a set of strings over this alphabet that is relevant to this application, and include a sentence or so justifying why this set is important and relevant.
3. Give one example of a string over this alphabet in this set and a string over this (same) alphabet not in this set, and explain why you chose these example strings.
4. Design a DFA that recognizes this language. Clearly draw and label the state diagram of this DFA and briefly justify why your design works by describing the role of each state of the DFA and relating it to a plain English description of the language you picked.
5. Use the construction in the proof that all regular languages are decidable (informally described in Theorem 4.1 in the book) to build a Turing machine that simulates your DFA. Hint: your Turing machine will have exactly two more states than your DFA. Draw the state diagram of your Turing machine.
6. For one of the strings from step 4., draw a representation of the computation of your Turing machine on this string. Remember that to describe the computation of a Turing machine, we need to include the contents of the tape, the state of the machine, and the location of the read/write head at each step in the computation. In class we've drawn pictures to represent the configuration of the machine at each step in a computation. You may do so or you may choose to describe these configurations in words.

Checklist for submission

- Solution typed or clearly hand-written/drawn with precise language and notation for all terms and complete, correct, and clear justification.
- Each of the six steps is complete and included in the PDF, with precise language and notation for all terms and complete, correct, and clear justification.

Your video: You may produce screencasts with any software you choose. One option is to record yourself with Zoom; a tutorial on how to use Zoom to record a screencast (courtesy of Prof. Joe Politz) is here:

`https://drive.google.com/open?id=1KROMAQuTCk4OzwrEFotlYSJJQdcG_GUU`.

The video that was produced from that recording session in Zoom is here:

`https://drive.google.com/open?id=1MxJN6CQcXqIbOekDYMxjh7mTt1TyRVMl`

Please send an email to the instructors (minnes@ucsd.edu) if you have concerns about the video / screencast components of this project or cannot complete projects in this style for some reason.