CSE 355: Module 2 Machine Design (Finite Automata)

Dr. Jamison W. Weber

Due Monday, September 2nd 2024, 11:59pm

In this individual assignment, you will formulate finite automata that recognize target languages using a Python programming interface. The programming interface enables you to formally define finite automata, check their validity, visualize them as state machines, and trace their computation on test strings. When you're satisfied with your finite automata, you will export and upload them to the corresponding Gradescope assignment for autograding. This will provide immediate, personalized feedback if there are any mistakes. You may revise and resubmit your code as many times as you'd like until the due date.

1 Target Languages

For each of the following problems, formulate a finite automaton that recognizes the target language.

Problem 1

$L_1 = \{w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ contains } \texttt{010} \text{ as a substring}\}$.

Problem 2

$L_2 = \{w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ does not contain } \texttt{100} \text{ as a substring}\}$.

Problem 3

$L_3 = \{w \in \{\texttt{a}, \texttt{b}\}^* \mid |w| < 4\}$.

Note. Recall that $|w|$ is the length of the string w, defined as the number of symbols in w.

Problem 4

$L_4 = \{w \in \{\texttt{a}, \texttt{b}, \texttt{c}\}^* \mid w \text{ has an even number of } \texttt{b}\text{'s}\}$.

Note. A string without any `b`'s at all has an even number of `b`'s: zero.

Problem 5

$L_5 = \{\varepsilon, \texttt{0}, \texttt{101}\}$, given the alphabet $\Sigma = \{\texttt{0}, \texttt{1}\}$.

Note. Recall that ε is the empty string.

Problem 6

$L_6 = \{w \in \{\texttt{1}\}^* \mid |w| \text{ is a multiple of 2 or 3}\}$.

Hint. Consider breaking the problem down into smaller subproblems and then combining your subsolutions using the product construction. Also, zero is a multiple of both 2 and 3.

Problem 7

$L_7 = \{w \in \{\texttt{a}, \texttt{b}\}^* \mid w \text{ has an even number of } \texttt{a}\text{'s, contains } \texttt{aab} \text{ as a substring, or both}\}$.

Hint. Consider breaking the problem down into smaller subproblems and then combining your subsolutions using the product construction.

2 Grading

The instructions in the following sections will show you how to use the programming interface, test your ideas, and ultimately export your solutions as a `.json` file. This file contains specifications of your finite automata, which Gradescope will instantiate and test to see if they correctly recognize the target languages.

Each problem (finite automaton) is worth 5 points. You get 1 point if the finite automaton is valid (i.e., it follows all the rules of a formal specification) and an additional 4 points if it recognizes the target language. If your finite automaton is invalid, Gradescope will tell you what the problem is. If your finite automaton recognizes a language other than the target one, Gradescope will output (1) example strings that your automaton recognizes but are outside the target language, and/or (2) example strings that are in the target language but your automaton does not recognize. This feedback should guide your resubmssions.

This assignment is worth 31 points, but with seven problems and 5 points per problem, there is the opportunity to earn up to 35 points total (up to 4 points of extra credit). This is designed such that you can submit valid finite automata for all seven problems but only get six of them correct to receive full credit.

3 Installation

In order to complete this and future Machine Design assignments, you will need to install both `python 3.10` (or a later version) and the `cse355-machine-design` package. You *do not* need Python programming knowledge to complete these assignments, as you will only use a small subset of the language's features.

3.1 Installing Python

If you already have Python installed on your system, check that its version is `3.10` or higher by running the `python --version` command in a terminal. Note that some systems expect the `python3` executable instead of `python`. If `python --version` doesn't work, try `python3 --version` instead. If both commands fail, you likely do not have Python installed.

To install Python, follow the instructions in this video tutorial covering both macOS and Windows installations: https://www.youtube.com/watch?v=YYXdXT2l-Gg. Remember that we are using Python 3.10 or later, not 3.6. The GUI is a bit out of date, but the process should remain pretty much the same.

If you'd like to follow written instructions instead, first visit https://www.python.org/downloads/ and download either the latest version (the big yellow download button at the top of the site) or Python 3.10.x (under "Looking for a specific release?"). This will download an installer to your system. At this point, you can install it as you would any other application. During the install process you may be asked if you would like to add `python` to your `PATH` variable. If this checkbox is present, *please ensure that it is checked*. After installing Python, open up a new terminal and check that `python --version` (see note above) produces an output that corresponds to the version you just installed. If you get a "command not found" error, either your install process did not properly add `python` to your `PATH` variable or Python was not installed.

3.2 Installing `cse355-machine-design`

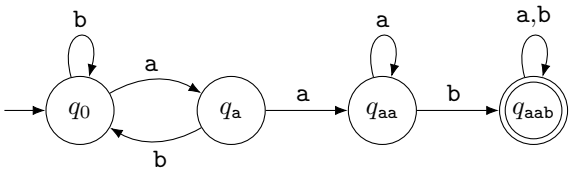
Once Python is installed, run the `pip install cse355-machine-design` command in a terminal to install the Machine Design package for this and future assignments. (If you used the `python3` executable before then you will use `pip3` instead of `pip` here.) If you ever need to update the package, run `pip install cse355-machine-design --upgrade`. For more details on the package—along with comprehensive documentation—please visit https://github.com/DaymudeLab/CSE355-MachineDesign.

4 Usage Guide

In this section, we demonstrate how to define and interact with finite automata using the `cse355-machine-design` package. Suppose that we are asked to formulate a finite automata recognizing the language

$$L = \{w \in \{\texttt{a}, \texttt{b}\}^* \mid w \text{ contains } \texttt{aab} \text{ as a substring}\}.$$

As you may recall from Lecture 3 of this module, the below finite automaton M recognizes L:



4.1 Defining a Finite Automaton

To start, download the template Python source file called `module2_machinedesign.py` from Canvas. The top of this file defines our package imports:

```
from cse355_machine_design import DFA, registry
```

Next, we have functions for defining the seven finite automata for this assignment (one per problem). Copy/paste one of these functions to add a new function for our example automaton, M:

```
def example():
    """
    L = {w in {a,b}* | w contains aab as a substring}
    """

    Q = {}
    Sigma = {}
    delta = {

    }
    q0 = ""
    F = {}

    return DFA(Q, Sigma, delta, q0, F)
```

The five variables you need to define (`Q`, `Sigma`, `delta`, `q0`, and `F`) are the five components in a finite automaton's formal definition $M = (Q, \Sigma, \delta, q_0, F)$. Finite automaton M has states $Q = \{q_0, q_{\texttt{a}}, q_{\texttt{aa}}, q_{\texttt{aab}}\}$, so in our Python function we write:

```
    Q = {"q_0", "q_a", "q_aa", "q_aab"}
```

Next, we define our alphabet. The input strings are defined over $\{\texttt{a}, \texttt{b}\}^*$, so:

```
    Sigma = {"a", "b"}
```

After this, we define our transition function δ. This is a little tricky, but can be thought of as a table where the first column defines the current state, the second column defines the symbol on the transition, and the third defines the next state to go to:

```
    delta = {
        ("q_0", "a"): "q_a",
        ("q_0", "b"): "q_0",
        ("q_a", "a"): "q_aa",
        ("q_a", "b"): "q_0",
        ("q_aa", "a"): "q_aa",
        ("q_aa", "b"): "q_aab",
        ("q_aab", "a"): "q_aab",
        ("q_aab", "b"): "q_aab"
    }
```


The start state of M is q_0, so:

```
    q0 = "q_0"
```

Finally, our set of accept states is:

```
    F = {"q_aab"}
```

The completed function is:

```
def example():
    """
    L = {w in {a,b}* | w contains aab as a substring}
    """

    Q = {"q_0", "q_a", "q_aa", "q_aab"}
    Sigma = {"a", "b"}
    delta = {
        ("q_0", "a"): "q_a",
        ("q_0", "b"): "q_0",
        ("q_a", "a"): "q_aa",
        ("q_a", "b"): "q_0",
        ("q_aa", "a"): "q_aa",
        ("q_aa", "b"): "q_aab",
        ("q_aab", "a"): "q_aab",
        ("q_aab", "b"): "q_aab"
    }
    q0 = "q_0"
    F = {"q_aab"}

    return DFA(Q, Sigma, delta, q0, F)
```

4.2 Interacting with Your Finite Automata

Now that we've defined the finite automaton M, we can put it to work. In a terminal, navigate to the directory containing your `module2_machinedesign.py` file and then run the `python` command. This will start an interactive session:

```
> python
Python 3.10.13
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Typing

```
>>> from module2_machinedesign import *
```

will import all definitions from your file into this interactive session. You can then set up one of your finite automata, like the example machine M that we defined:

```
>>> example_fa = example()
```

Our programming interface provides tools for investigating and testing your finite automata. First, you can open a state diagram of your finite automaton in your web browser using:

```
>>> example_fa.display_state_diagram()
```

Second, you can evaluate whether an input string is accepted (`True`) or rejected (`False`) by passing it to the `evaluate` function:


```
>>> example_fa.evaluate("abaaabb")
True
>>> example_fa.evaluate("abababa")
False
```

If you want to see a detailed trace of the finite automaton's computation instead of just the final accept/reject result, add the `enable_trace=1` parameter:

```
>>> example_fa.evaluate("abaaabb", enable_trace=1)
Starting at state "q_0"
Reading input "a". This causes us to transition from "q_0" to "q_a"
Reading input "b". This causes us to transition from "q_a" to "q_0"
Reading input "a". This causes us to transition from "q_0" to "q_a"
Reading input "a". This causes us to transition from "q_a" to "q_aa"
Reading input "a". This causes us to transition from "q_aa" to "q_aa"
Reading input "b". This causes us to transition from "q_aa" to "q_aab"
Reading input "b". This causes us to transition from "q_aab" to "q_aab"
Finished reading input. We are now in state "q_aab". This is a final state, so we accept
True
```

4.3 Submitting Your Finite Automata

At the bottom of the template file, you will see the following code:

```
if __name__ == "__main__":
    problem1().submit_as_answer(1)
    problem2().submit_as_answer(2)
    problem3().submit_as_answer(3)
    problem4().submit_as_answer(4)
    problem5().submit_as_answer(5)
    problem6().submit_as_answer(6)
    problem7().submit_as_answer(7)
    registry.export_submissions()
```

Don't change this code! Each `problem#()` function is building and returning a finite automaton based on how you filled out those functions above. The `submit_as_answer` function is recording those finite automata as your answers to the corresponding problems. It is *very important* that the parameter to the `submit_as_answer` function matches the problem number! If these differ, Gradescope won't be able to match up your answers to its grading code and you won't receive any points.

When you're ready to submit your answers, run `python module2_machinedesign.py` on the command line in the directory of your assignment file. This will create a `submissions.json` file that you upload to Gradescope to be graded.

