Term Project

San José State University
Department of Computer Science

CS 154: Formal Languages and Computability
Fall 2017

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu

Objective

To design and implement a universal TM that can run any DFA.

Project Description

We are going to design and implement a universal TM whose input is the definition of an arbitrary DFA called M and the M's arbitrary input string called w.

The TM feeds w to the M and simulates M's entire operations until M halts. Then, it shows "A" (without the quotes) if M accepts w or "R" if M rejects w.

How can we put a DFA's definition and its input string on the tape of the TM?

As we've learned so far, the input of TMs (and other automata) are strings. w is already a string, so, we need to describe M by a string. Describing a machine as a string is called "encoding" and we'll explain it in the next section. We'll see later that it would be much easier if we encode the w as well.

To explain everything clearly, we'll take an example and explain the whole process through it.

Example

Let M be the following DFA and let w be w = bba.

M can be defined mathematically as M = $(Q, \Sigma, \delta, q_0, F)$, where:

$Q = \{q_2, q_5, q_9\}$, $\Sigma = \{a, b\}$, $q_0 = q_2$, $F = \{q_2, q_5\}$, and the transition function is:

$\delta(q_2, b) = q_2$

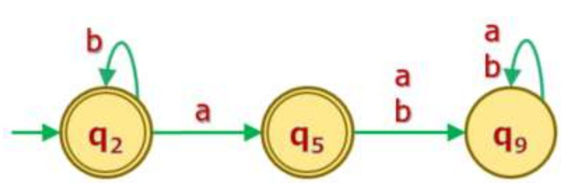
$\delta(q_2, a) = q_5$

$\delta(q_5, b) = q_9$

$\delta(q_5, a) = q_9$

$\delta(q_9, b) = q_9$

$\delta(q_9, a) = q_9$



How to Encode DFAs

We shall encode all elements of the DFA and w by **unary numbers** as follows.

Q = {q_2, q_5, q_9}

The first element of Q is encoded as 1, the second one as 11 and so forth. So, the encoded version of Q would be: Q = {1, 11, 111}

q_0

We always put the **initial state as the first element of Q**. So, q_0 (e.g. q_2 in the example) is **always** encoded as 1.

Σ = {a, b}

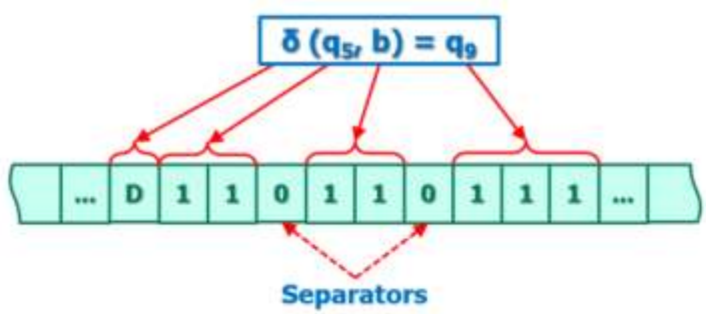
The first element of Σ is encoded as 1, the second one as 11 and so forth. So, the encoded version of Σ would be: Σ = {1, 11}

F = {q_2, q_5}

The set of final states follows the same codes of Q. So, the encoded version of F would be F = {1, 11}.

δ (q_i, x) = q_j

The elements of sub-rules are encoded by the same codes of Q and Σ and are formatted as the following figure shows. We use '0' (zero) as the separator between the elements.



Note that q_5 and b have the same code (i.e. 11), but their locations in the string give them different meaning.

All sub-rules of the example have been encoded in the following table.

Sub-Rule	Encode String
$\delta\ (q_2, b) = q_2$	D101101
$\delta\ (q_2, a) = q_5$	D101011
$\delta\ (q_5, b) = q_9$	D110110111
$\delta\ (q_5, a) = q_9$	D11010111
$\delta\ (q_9, b) = q_9$	D1110110111
$\delta\ (q_9, a) = q_9$	D111010111

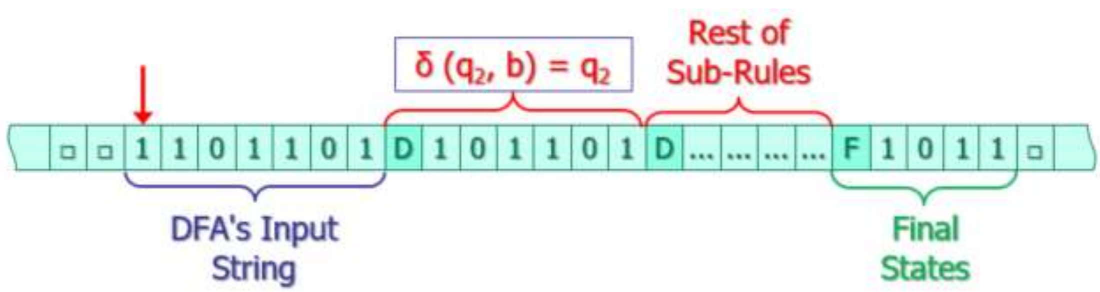
DFA's Input String w = bba

The symbols of the input string are encoded by the codes of Σ and are separated by '0' (zero). So, the encoded version of w would be: w = 1101101

Now, let's put all together and construct the TM's input string that contains the DFA's desciption and its input string.

Encoded DFA and w On TM's Tape

The following figure shows the encoded DFA in the example and w, its input string, on the TM's tape. In fact, this string would be the TM's input string.



Notes

1. The order of the elements of Q does not matter. The only restriction would be the first element that must be the q_0 of the DFA.
2. The order of the elements of Σ does not matter.
3. We don't need to put Q and Σ in the TM's input string because we just need to use their codes in δ, w, and F.

4. We don't need to put q_0 in the TM's input string because it is fixed for all DFAs. In other words, it is always the first element of Q and always is encoded as 1.
5. The order of sub-rules does not matter.
6. The states in F are separated by '0' (zero) and their **order does not matter**. We might have no final state. In this case, there is just TM's **blank**.
7. **To feed λ**, just put nothing in that place. In this case, the TM's input string starts with 'D' of the first sub-rule.

So, if we apply all the rules we mentioned, the final TM's input string for the example would be:

1101101D101101D101011D110110111D11010111D1110110111D111010111F1011

And as usual, when the TM starts, the read-write head is located on the first symbol of the string.

Your TM is supposed to use this string and run the DFA against the provided w (bba in the example) and show the output.

Note that this is just an example and your TM should be able to run any arbitrary DFA against any arbitrary w.

TM's Output

If the DFA accepts w, the TM shows 'A' (as Accept) and if it rejects w, the TM shows 'R' (as Reject). For the DFA and w of our example, your TM is supposed to show 'A' (**without the quotes** of course.)

Please refer to my lecture notes and/or JFLAP's documents for **how JFLAP shows outputs**.

Technical Notes

1. We assume that the input string of the TM is 100% correct. It means, M and w are encoded and formatted correctly. Therefore, your TM is not supposed to have any error checking or error reporting.
2. You might use extra features of JFLAP such as: "S" (= stay option), block feature, variable assignments, and JFLAP's special characters '!' and '~'. These are great features that tremendously facilitate the design process and make life easier. For more information, please refer to the JFLAP's documentations and tutorials.
3. Before implementing and testing your design, make the following changes in JFLAP's preferences:
   In Turing Machine Preferences: uncheck "Accept by Halting" and check the other options.

4. Test your Turing machine as a **transducer** option of JFLAP.
5. Organize your design in such a way that it shows different modules clearly. Also, document very briefly your design by using **JFLAP notes**. These are for maintainability purpose and it won't affect your grade.
6. Be careful when you work with JFLAP's block feature. It is a buggy software specially when saving a block. So, always have a backup of your current work before modifying it.

Rubrics

- I'll test your design with 20 test cases (different DFAs against different input strings) and you'll get +10 for every success pass (**200 points total**).
- If your code is not valid (e.g. there is no initial state, it is implemented by JFLAP 8, or so forth) you'll get 0 but you'd have chance to resubmit it with -20% penalty.
- You'll get -10 for wrong filename!
- Note that if you resubmit your assignment several times, Canvas adds a number at the end of your file name. **I won't consider that number as the file name**.

What You Submit

1. Design and test your program by the provided JFLAP in Canvas.
2. Save it as: Team_CourseSection_TeamNumber.jff
   (e.g.: Team_2_6.jff for team number 6 in section 2. The word "Team" is constant for all teams.)
3. Upload it in the Canvas before the due date.

General Notes

- **Always read the requirements at least 10 times!** An inaccurate computer scientist is unacceptable!
- This is a **team-based project**. So, the members of a team can share all information about the project but you are **NOT allowed to share** the info with other teams.
- The only thing that you can share with other teams is your test cases via Canvas discussion.
- Always make sure that you have the latest version of this document. Sometimes, based on your questions and feedback, I need to add some **clarifications**. If there is a new version, it will be announced in the class and via the Canvas.
- After submitting your work, always download it and test it to make sure whether the process of submission was fine.
- For **late submission policy**, please refer to the greensheet.
- If there is any question or concern, please open a discussion in Canvas.

Working with JFLAP

- Always have separate file for each module (aka 'block').
- If module A has a problem and you need to change it, change its original file and save it. Then, if module B is using module A, you need to re-inject module A in the module B and save B again.
- Be careful about these procedures and always have a working backup of every modules. It would be safer if you can use **version control** for this project.

Hints about Teams

The roles you'd need for your team:

1. Project manager
    a. Breaking down the whole project into smaller activities and tasks
    b. Scheduling the tasks
    c. Controlling the schedule and making sure that the project is on time.
2. Architect
    a. Designing the top level of the modules
    b. Integrating the modules and testing
3. Developer
    a. Breaking down the top-level modules into lower level
    b. Implementing and unit-testing the smaller modules
    c. Integrating the smaller modules into higher level and integration-testing
4. Tester
    a. Testing every module and trying to break it
    b. Testing the entire TM

Everybody need to have one role but note that everybody should be developer. So, everybody should pick one role plus developing.