Term Project

San José State University
Department of Computer Science

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu

CS 154: Formal Languages and Computability
Spring 2018

Objective

To design and implement a "Universal Turing Machine (UTM)" that can run any arbitrary TMs against any arbitrary string w.

Suggestion

Before implementing this project, you are highly recommended to see the project of the previous semester that was simulating DFAs by a universal TM. The requirement, the selected implementation done by a team, and test data can be found in "**Canvas -> Files -> Project -> F17 Proj**" folder.

Please note that, to understand the solution, you'd need to know the concept of "block" in JFLAP.

Project Description

We are going to design and implement a "Universal Turing Machine (UTM)" whose input is the definition of an arbitrary TM called M and an arbitrary input string of M, called w.

The UTM feeds w to M and simulates M's entire operations against w until M halts. Then it shows 'A' (**without the quotes**) if M halts in an accepting state (accepts w), or 'R' if M halts in a non-accepting state (rejects w).

Obviously, if M falls in an infinite-loop, then UTM would fall automatically in an infinite-loop too.

How can we put a TM's definition on the tape of UTM?

As we know, the input of TMs (and other automata) are strings. Therefore, we need to describe M by a string.

Describing any object as a string is called "encoding". Next section is devoted to explaining how to encode objects like transition graphs. The idea can be extended to trees, graphs, and any other objects.

Encoding Objects

We'd better to explain the whole process through an example.

Example

Let M be the following TM and let w = ab.

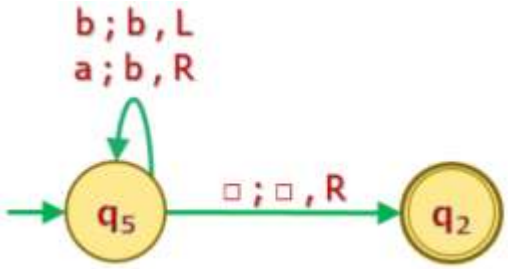
M can be defined mathematically by M = (Q, Σ, Γ, δ, q_0, □, F), where:

Q = {q_5, q_2}, Σ = {a, b}, Γ = {□, a, b}, q_0 = q_5, F = {q_2}, and the transition function δ is:

δ (q_5, a) = (q_5, b, R)

δ (q_5, b) = (q_5, b, L)

δ (q_5, □) = (q_2, □, R)



We shall encode all elements of M and w by **unary numbers** as follows:

Q = {q_5, q_2}

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

q_0

We always put the **initial state as the first element of Q**. So, q_0 (e.g. q_5 in this example) is **always** encoded as 1.

Γ = {□, a, b}

The first element of Γ is encoded as 1, the second one as 11 and so forth. So, the encoded value of Γ would be: Γ = {1, 11, 111}. Note that since Σ ⊆ Γ – {□}, so, we don't need to encode Σ independently.

F = {q_2}

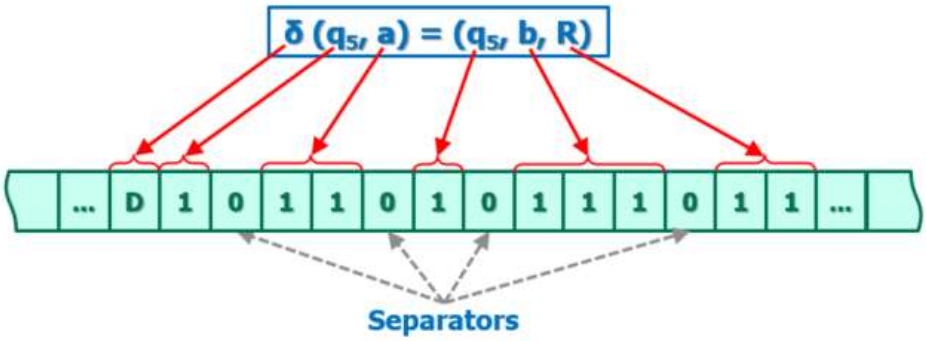
The set of final states is encoded by using the same codes of Q. So, the encoded value of F would be F = {11}.

δ (q_i, x) = (q_j, y, R)

We encode L (left) movement as 1 and R (right) as 11.

The other elements of sub-rules are encoded by the same codes of Q and Γ and are formatted as the following schematic shows.

We use '0' (zero) as the separator between the elements.



Note that in this example, q_2, R, and 'a' have the same code (i.e. 11), but their locations in the string give them different meaning.

The following table shows the encoded values of all sub-rules of the example.

Sub-Rule	Encode String
δ (q_5, a) = (q_5, b, R)	D1011010111011
δ (q_5, b) = (q_5, b, L)	D1011101011101
δ (q_5, □) = (q_2, □, R)	D10101101011

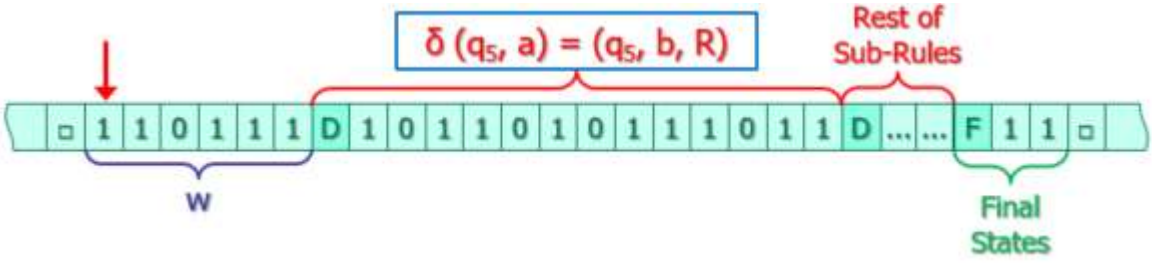
Encoding M's Input String w = ab

The symbols of the w are encoded by the codes of Γ and are separated by '0' (zero). So, the encoded value of w would be: w = 110111

Now, let's put all together and construct the UTM's input string that contains the M's description and its input string w.

Encoded Values of M and w On UTM's Tape

The following schematic shows the partially encoded values of M and w, on UTM's tape. In fact, this string would be the UTM's input string.



Encoding Notes

1. The order of the elements of Q does not matter. The only restriction is the first element that must be always M's initial state q_0.
2. The order of the elements of Γ does not matter.
3. The order of sub-rules does not matter.
4. There might be zero, one or more final states. In the case of zero, there is only blank after F. In the case of more than one, the codes of the F's are separated by '0' (zero) and their **order does not matter**.
5. If w is λ, then nothing will be put in the w place. In that case, the UTM's input string starts with 'D' of the first sub-rule.

So, if we apply all the above rules, the UTM's input string for the example would be:

110111D1011010111011D1011101011101D10101101011F11

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

Your UTM is supposed to use this string and run M against w (ab in the example) and show the appropriate output.

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

TM's Output

If M accepts w, the UTM shows 'A' (as Accept) and if it rejects w, the UTM shows 'R' (as Reject). For M and w of our example, your UTM 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 UTM is 100% correct. It means, M and w are encoded and formatted correctly. Therefore, **your UTM is not supposed to have any error checking or error reporting**.
2. You are highly recommended to use extra features of JFLAP such as: "S" (= stay option), block feature, 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 UTM in **transducer** mode 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's notes**. These are for maintainability purpose and **they 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. For more information about this, please refer to the section "working with JFLAP".

Rubrics

- I'll test your design with 20 test cases (different TMs 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 project 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. Note that your final submission is **only one file**.
2. Save it as: Team_CourseSection_TeamNumber.jff
   (e.g.: Team_2_15.jff is for team number 15 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 via 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 ambiguity, question, and/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.