Term Project

CS 154: Formal Languages and Computability
Fall 2018

San José State University
Department of Computer Science

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu

Objective

To design and implement a "Universal Turing Machine (UTM)" that can run any arbitrary DPDA (**deterministic pushdown automata**) against any arbitrary string w.

Note that DPDA has been chosen over NPDA because implementing parallel processing needs more time.

Suggestion

Before implementing this project, you are highly recommended to **see the last year term project** that was simulating DFAs by a TM.

You can find "**F17 Selected Proj.zip**" file in "**Canvas -> Files -> Project** " folder. This file contains: term project requirement, a selected implementation done by a team, test data, and **a version of JFLAP that is just for testing big TMs**.

Please note that, to understand the solution, you'd need to know about the concept of "block" and some characters such as '!' and '~' in JFLAP.

Project Description

You are going to design and implement a "Universal Turing Machine (UTM)" whose input is the definition of an arbitrary DPDA 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).

How can we put a DPDA'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 of a machine.

Of course, the idea can be extended to other data structures such as trees, graphs, and so forth.

Encoding Objects

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

Example

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

As we learned, M can be defined mathematically by M = (Q, Σ, Γ, δ, q_0, Z, F), where:

Q = {q_0, q_1, q_2, q_3}, Σ = {a, b}, Z is the special symbol called Stack's start symbol,
Γ = {Z, a, b}, q_0 = q_0, F = {q_0, q_3}, and the transition function δ is:

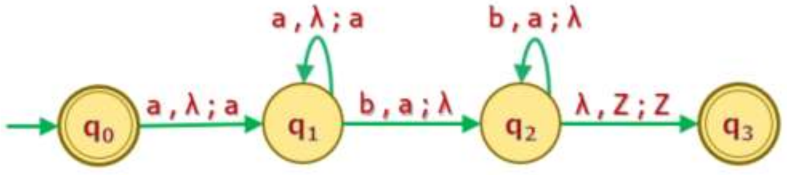
δ (q_0, a, λ) = (q_1, a)

δ (q_1, a, λ) = (q_1, a)

δ (q_1, b, a) = (q_2, λ)

δ (q_2, b, a) = (q_2, λ)

δ (q_2, λ, Z) = (q_3, Z)



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

Q = {q_0, q_1, q_2, q_3}

The first element of Q is encoded as 1, the second one as 11, and so forth. So, the encoded value of Q for this example would be: Q = {1, 11, 111, 1111}

q_0

We always put the **initial state as the first element of Q**. So, q_0 (e.g. q_0 in this 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 value of Σ for this example would be: Σ = {1, 11}.

Γ = {Z, 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 for this example would be: Γ = {1, 11, 111}.

Notes for Γ and Σ

1. Since Z is always a member of Γ, so, we put it always as the first element of Γ and encode it as 1.

2. Γ and Σ are independent and are encoded independently even though they have common symbols. For example, 'a' is encoded as 1 for Σ but is encoded as 11 for Γ.
3. There is another symbol that belongs to neither Σ nor Γ and that is **λ**.
   For simplicity, **we encode it as** '**m**'.

F = {q_0, q_3}

The elements of the set of final states are encoded by using the same codes of Q. So, the encoded value of F for this example would be F = {1, 1111}.

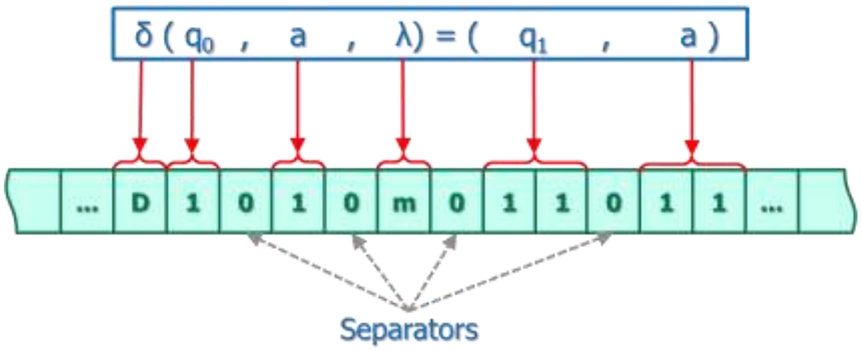
δ (q_i, a, x) = (q_j, u)

We encode **u** ∊ **Γ*** by using the codes of Γ and delimit the symbols by '**0**' (zero).

For example, if u = abb, then it would be encoded as: 11**0**111**0**111

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

We use '0' (zero) as the delimiter.



Note that in this example, q_1, b ∊ Σ, and a ∊ Γ, all 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_0, a, λ) = (q_1, a)	D1010m011011
δ (q_1, a, λ) = (q_1, a)	D11010m011011
δ (q_1, b, a) = (q_2, λ)	D110110110111 0m
δ (q_2, b, a) = (q_2, λ)	D11101101101110m
δ (q_2, λ, Z) = (q_3, Z)	D1110m010111101

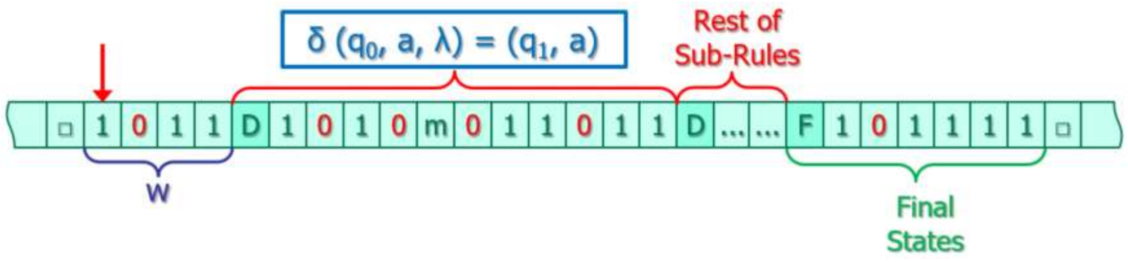
Encoding M's Input String w

The symbols of the w (w = ab in this example) are encoded by the codes of Σ and are delimited by '0' (zero). For example, w = **ab** is encoded as **1011**.

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 figure shows 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, as we mentioned earlier, is the first element of Q that must be always M's initial state q_0 and is encoded as 1.
2. The order of the elements of Γ and Σ does not matter. The only restriction, as we mentioned earlier, is the first element of Γ that is always Z and is encoded as 1.
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, the input string of M, 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 above rules, the UTM's input string for the example would be:

1011D1010m011011D11010m011011D1101101101110mD1110110110111 0mD1110m010111101F101111

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 whatever we explained was just an example. **Your UTM should be able to run any arbitrary DPDA against any legal w**.

DPDA'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 detection 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. I'll review them quickly in the class, but 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 only and **do NOT 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" of this document.

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.
4. **One submission per team is enough.**

Rubrics

- I'll test your design with 20 test cases (different DPDAs 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**.

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** any info with other teams.
- The only thing that you can share with other teams is your test cases via Canvas discussion. **If your test cases are good enough, then I might use them for testing your designs as well.**
- 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 that 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 modify 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 a **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 DPDA

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