Term Project

CS 154: Formal Languages and Computability
Spring 2019

San José State University
Department of Computer Science

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu

"A good developer always reads the requirements at least 10 times! *Ahmad Y"*

Objective

To design and implement a **Universal Turing Machine (UTM)**[1] that simulates the operations of any arbitrary **digital circuits** against any arbitrary binary inputs.

Before Starting

Before implementing this project, you are highly recommended to **see one of the previous term projects** that was simulating DFAs by a UTM.

You can find "**PreviousSelectedProj.zip**" file in "**Canvas -> Files -> Project** " folder. This file contains: the project requirements, a selected implementation done by a team, test data, and a JFLAP to run this project[2]. The zip file is compressed by 7-zip and the password is ahmady.

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

Project Description

You are going to design and implement a Universal Turing Machine (UTM) whose input is the **definition of an arbitrary digital circuit called DC** and an **arbitrary input binary string called w**.

The UTM applies the given w to the DC and simulates DC's entire operations. Then it shows the result that could be 0 or 1.

What is a digital circuit's definition?

As we know, the input of TMs (and other automata) are strings. Therefore, we need to describe DC by a string to be able to put it on the UTM's tape.

Describing any kind of objects (of course except strings) as a string is called "encoding". There are many ways to encode digital circuits and we engaged one way that will be explained in the next section.

This idea of encoding can be extended to other data structures such as trees, graphs, matrices, functions, transition graphs of other automata, and so forth.

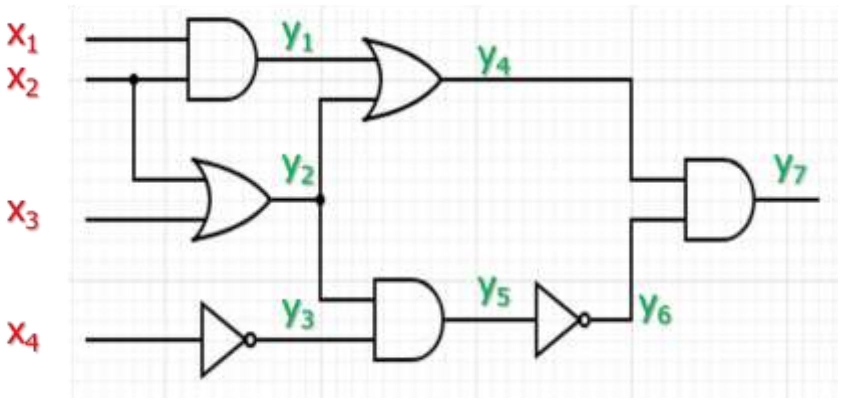
[1] Universal Turing Machine (UTM) is a TM that simulates other TMs but here we are using it as a general name for a TM that simulates any other objects such as digital circuits.
[2] This is a special **version 7.0** JFLAP compiled by me. This version does not bug you after every 500 steps. Note that this JFLAP cannot be used in your term project. I'll provide one of these later.

Encoding Digital Circuits

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

Example

Let DC be the following digital circuit and let w = 1011 be the input values of $x_1\, x_2\, x_3\, x_4$.



We can define the outputs of the gates by using boolean operations as:

$y_1 = x_1 \,.\, x_2$

$y_2 = x_2 + x_3$

$y_3 = \overline{x_4}$

$y_4 = y_1 + y_2$

$y_5 = y_2 \,.\, y_3$

$y_6 = \overline{y_5}$

$y_7 = y_4 \,.\, y_6$

Where '**.**' (dot) is **boolean AND**, '+' (plus) is **boolean OR**, and $\overline{x_4}$ (bar) is **boolean NOT**. In fact, the above circuit is the boolean function:

$y_7 = f(x_1 , x_2 , x_3 , x_4)$

To put this circuit on a TM's tape, we need to encode it as a string. We shall encode the above boolean equations as shown in the following table:

Index	Equation	Encoded
1	$y_1 = x_1 . x_2$	Ax#xx
2	$y_2 = x_2 + x_3$	Oxx#xxx
3	$y_3 = \overline{x_4}$	Nxxxx
4	$y_4 = y_1 + y_2$	Oy#yy
5	$y_5 = y_2 . y_3$	Ayy#yyy
6	$y_6 = \overline{y_5}$	Nyyyyy
7	$y_7 = y_4 . y_6$	Ayyyy#yyyyyy

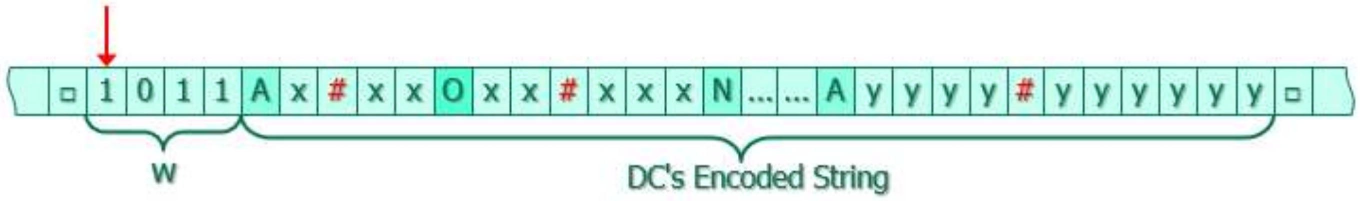
Encoding Description

1. **We use prefix notation** instead of infix for the boolean operations.
2. 'A' means AND, 'O' means OR, and 'N' means NOT.
3. We use **unary number style** for the indices. For example, x_3 is encoded as xxx, y_5 is encoded as yyyyy, and so forth.
4. We don't need to encode the left-hand y's and the equal sign because it's redundant. It means, for $y_1 = x_1 . x_2$ we don't need to encode "$y_1 =$" part.
5. Consider the above function sub-rules as the assignment statements in a regular programming language like C. Therefore, **every y's should be calculated before it can be used on the right-hand side of the statements bellow it**.
6. Each input of the circuit can be 0 or 1 and all of them together makes a binary number that the most-significant-bit (MSB) will be the top input and the least-significant-bit (LSB) will be the bottom one (i.e. $x_1 x_2 x_3 x_4$).
7. The input to the circuit does not need to be encoded because it's already been encoded as binary. (e.g. 1011 for the above example).
8. The sub-outputs, y's, are numbered from the top to the bottom and from left to the right.

Encoded DC and w On UTM's Tape

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

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


□ 1 0 1 1 A x # x x O x x # x x x N … … A y y y y # y y y y y y □
W
DC's Encoded String

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

1011Ax#xxOxx#xxxNxxxxOy#yyAyy#yyyNyyyyyAyyyy#yyyyyy

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

Your UTM is supposed to use this string and calculate all y's and finally show the last y's value.

Note that the above explanation is just an example. **Your UTM should be able to simulate any arbitrary digital circuit against any binary input**.

Output

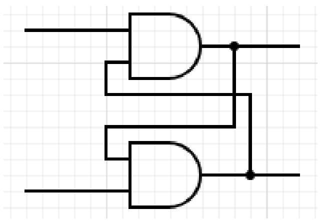
At the end of the operations, when your UTM halts, the only thing that matters is the output of the circuit against the given input that could be 0 or 1. Therefore, the other contents of the tape do not matter at all.

For the above example, the input is 1011, the output is y_7, and the result of the calculation that should be shown by the JFLAP is 1.

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

Circuits Notes

1. The circuits might have any number of inputs, but it always has one output.
2. The circuits are always flat and there won't be any feedback from the output of a gate to itself or so. For example, the following circuit is out of this project's scope:



Technical Notes

1. **We assume that the input string of UTM is 100% correct**. It means, the DC and w are encoded and formatted correctly. Therefore, **your UTM is not supposed to have any error detection or error reporting**.
2. **You must use block feature of JFLAP.** This feature is located in "**Turing Machine With Building Blocks**" button of JFLAP. I'll have a quick demo about this.
3. Organize your design in such a way that it shows different modules clearly.
4. Test your UTM in **transducer** mode.
5. Document very briefly your design by using **JFLAP's notes**. This is only for maintainability purpose and **does NOT affect your grade**.

JFLAP Notes

1. Be careful when you work with JFLAP's block feature. **It is a buggy** software, **specially when editing and saving a block inside another 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.
2. Before implementing and testing your code, **make the following changes in JFLAP's preferences**:
   In the Turing Machine Preferences: uncheck "Accept by Halting" and check the other options. JFLAP creates the XML file jflapPreferences.xml.
3. You are highly recommended to use **extra features of JFLAP** such as: "S" (stay option), 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.

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 digital circuits against different inputs) 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 7.0, 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 developer is unacceptable!
- **After submitting your work, always download it and test it to make sure that the process of submission was fine.**
- 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 info that you can share with other teams is your test cases via Canvas discussion. **I might use them for grading If your test cases are good enough!**
- 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.
- 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 much safer if you use a version control for this project**.

Hints about Teams Configuration

The roles you'd need for your team:

1. Project manager (usually is the assigned leader of the team)
    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.
    d. Submitting the final code
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 UTM

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