Practice lessons of Theory of automata and Formal languages

Prof. Karl Thurnhofer Hemsi
Computer Science and Computer Language Department
University of Málaga

November 26, 2020

Practice 3

Turing Machine, recursive functions and WHILE language

3.1 Turing Machine (TM)

First, we will use JFLAP to learn how to implement a Turing Machine. Unlike the representation seen in class, in this software the TM's are generated as a graph, i.e., as a finite automaton.

- Each node correspond to a state of the MT.
- Each transition of a node is characterized by a symbol of the alphabet, so we are representing the transition function.
- In addition, in each transition the instruction is indicated, that is, it is necessary to indicate the new symbol that is going to be replaced in the tape, and the direction in which the reading head is going to move: R (right), L (left) or S (stay).
- The reading head is initially located in the first symbol of the chain to be processed, that is, at the beginning of the chain.
- The initial and terminal configurations are represented as in the case of FA. A string is accepted by the MT when it reaches a final state.

Note that there is a small difference (in addition to the initial position of the reading head) with respect to the definition seen in class: in JFLAP we can replace a symbol of the alphabet and move the reading head in the same instruction. This implies a small extension of the TM initially defined. We can restrict it "manually" by executing S when we make a substitution, and keeping the same symbol when we want to move R or L.

Exercises

1. Run JFLAP and select the 'Turing Machine' option.
2. Open the file "ExampleMT.jff".
3. Analyze the route $q0 \to q1 \to q2 \to q5 \to q6 \to q0$. What does the TM do?
4. And in $q0 \to q3 \to q4 \to q5 \to q6 \to q0$?
5. Test the string $aab\$aab$ using the option 'Input → Step ...'. The status $q0$ will appear shaded, and the tape, the current state and the position of the reading head will appear in the lower box.

6. Press the 'Step' button and see what happens. When we go back to state $q0$, what happens? Do you agree with what you have answered in question 3?

7. Continue until you reach state $q8$. Has the chain been accepted?

8. Try now with the string $aab\$aba$. Is this chain accepted? What happened?

9. What is the method of operation of this TM?

10. Test different chains using the option 'Input $\rightarrow$ Multiple Run' and try to find out which language accepts this TM.

3.2 Recursive functions

This section aims to revise the definition of the recursive functions seen in class. In the *tafluma* Bitbucket's project can be found two Octave's functions to evaluate and express recursive functions previously defined in a database file, which is called "recursivefunctions".

- evalrecfunction.m: given a recursive funciton and its arguments, evaluates the expression and outputs the result.
- recursiveexpression.m: given the database file and the name of the function, this function outputs the recursive expression.

Exercises

1. Obtain the recursive expression of the function `constant`^2_3.
2. How many arguments has the previous function?
3. Evaluate this function with different values and study the behaviour.
4. Which mathematical function represents?
5. Evaluate the recursive function $\mu[<\pi^1_1(\sigma)|\pi^3_2(\pi^1_1(\theta),\sigma,\pi^1_1(\theta))>]$. What happens?

3.3 WHILE language

For this last part of the practice we will use a WHILE language compiler-debugger developed in JAVA and created by a student of our School as an End-of-Course Project. Download the program from the Virtual Campus and execute it.

The operation is simple and similar to other programming environments (such as Eclipse). On the left we have the editor, below the console where the execution trace appears, above we have a box in which we can define conditional breakpoints and on the right we can add "watches" to observe the value of the variables during the execution of the program.

This compiler accepts the same WHILE language as we have seen in class, being $Q = (n, s)$ a While program with n input variables. The total number of variables (included auxiliaries) are represented by the parameter p. The compiler also supports an extended version, the expanded WHILE language, which includes more statements for assigning constants, addition and subtraction of any value, control statement of the *while* loop, being able to compare with any value, and even the definition of macros or subprograms.

Exercises

1. Open the file "ExampleWHILE.txt".
2. On the right there is a panel with buttons. We will press the first of them (green arrow) to load our program.
3. A small box will appear in which we will have to indicate the values of our input variables. Enter the values $X1 = 5$ and $X2 = 4$.
4. Next, several more buttons are enabled in the right panel. The 'bug' is used to run our program completely, while the 'step' buttons are used to go line by line. Use the latter to see the execution of the program step by step. In the lower console, configurations and the computation appear for each step we perform.
5. What is the initial configuration of the program? And the final one?
6. Calculate $next_Q(1, 3, 3), cal_Q(3, 3, 6), T_Q(3, 3), F_Q(3, 3)$.
7. Calculate $F_Q(4, 5), F_Q(5, 3), F_Q(2, 1), F_Q(3, 7)$. What predicate defines this function?

Activities

1. Define the TM solution of exercise 3.4 of the problem list and test its correct behaviour.
2. Define a recursive function for the sum of three values.
3. Implement a WHILE program that computes the sum of three values. You must use an auxiliary variable that accumulates the result of the sum.