CS581 Theory of Computation: Homework #3

Due on February 10 2016 at 2:00pm

Harry H. Porter Winter 2016

Konstantin Macarenco

Problem 1

Provide a formal description of Turing Machines.

Solution:
A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite setns and.

1. Q is the set of states,
2. Σ is the input alphabet not containing the **blank symbol** $\sqcup$
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is t he transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.

Problem 2

Describe informally (no more than half a page) the operation of a Turing Machine.

Solution:
Turing machine is a theoretical computation model similar to to finate automation but with unlimited and unrestricted memory. It accurately models a general purpose computer.
Turing machine consists of infinite tape (unlimited memory), and a tape head that can move left and righ, scan and write symbols from/to the tape. Initially input is placed on tape, with the rest of the tape containing blanks, when computation starts with head positioned at the beginning of the input. During computation the head can scan the symbol under the head, replace it with another symbol and move left (can't pass start point) or right, one symbol at a time. When computation is over, TM will be in reject or accept state, with the resulting output written on the tape. For some strings TM will never reach accept or reject sate it will instead loop forever.

Problem 3

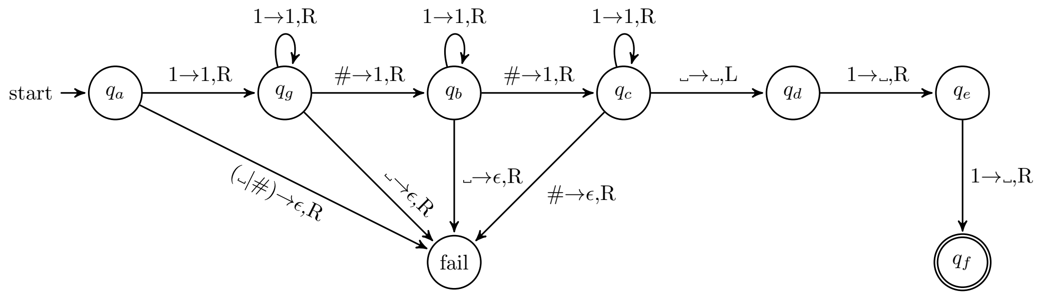
Design a Turning Machine that takes 3 numbers in unary representation and adds them together, leaving the result on the tape. (Unary representation:5 in unary is 11111). Assume the three numbers are separated by the # symbol. For example, the problem $3 + 4 + 2$ would be represented on the tape as: 111#1111#11 the machine should accept with the following string on the tape: 111111111. Give your machine in graph notation, in the style of Figure 3.8.

Solution:
The problem doesn't specify if we are allowed to have empty strings, i.e. string in form ##1111. This machine is created under assumption that empty strings are not allowed.

Machine M that recognizes this language is simple, since we now upfront that 3 numbers are added separated by two #. Machine M works as follows:

"On input w:

1. scan first symbol, if it is not 1, reject, otherwise replace it with 1 and move right.
2. scan next symbol, if 1 found, replace with 1 move right and repeat, if it is #, replace it with 1 goto next, reject on blank
3. scan next symbol, if 1 found, replace with 1 move right and repeat, if it is #, replace it with 1 goto next, reject on blank
4. scan next symbol, if 1 found, replace with 1 move right and repeat, if it is # reject, if blank found goto next
5. move two positions left, and erase symbols every move (replace with blanks). Accept.
6.



Problem 4

What is a Decidable Language?

Solution:
A language L is decidable (or Turing decidable) if there exists a Turing machine M such that on input x, M accepts if $x \in L$, and *rejects* otherwise.

Problem 5

What is a Turing-Recognizable Language?

Solution:
A language L is recognizable (or Turing recognizable) if there exists a Turing machine M such that on input x, M accepts if $x \in L$, but may either reject or loop forever otherwise.

Problem 6

What is a Recursively Enumerable Language?

Solution:
Same as Turing-recongnizable.

Problem 7

State the Church-Turing Thesis.

Solution:
Church-Turing Thesis - algorithm/problem is computable if it's computable by a Turing Machine/Lambda Calculus.

Problem 3.6

In Theorem 3.21, we showed that a language is Turing-recognizable iff some enumerator enumerates it. Why didn't we use the following simpler algorithm for the forward direction of the proof? As before, $s_1, s_2, \cdots$ is a list of strings in Σ^*

Solution:
E = "Ignore the input.

1. Repeat the following for $i = 1, 2, 3, \cdots$.
2. Run M on s_i.
3. If it accepts, print ou t s_i"

The problem with this definition is that if on step 2 M loops on some input s_j, then E will fail to check any input after, in this case E will fail to enumerate the language.

Problem 3.8

Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet $\{0, 1\}$

Problem 3.8 (b)

$\{w|w$ contains twice as many 0s as 1s$\}$

Solution:

1. Scan the tape for the first 0 that has not been marked, if none found, goto the last step, otherwise mark it.
2. Move to the next unmarked 0, if none reject, otherwise mark it and move back to the start of the input.
3. Scan for the first unmarked 1, if none reject, otherwise mark it.
4. Move the head to the beginning of the input, and goto step 1.
5. Move the head to the beginning of the input, and scan for unmarked 1s, if none accept, otherwise reject.

Problem 3.9

Let a k-PDA be a pushdown automaton that has k stacks. Thus a 0-PDA is an NFA and a 1-PDA is a conventional PDA. You already know that 1-PDAs are more powerful (recognize a larger class of languages) htan 0-PDAs.

a. Show that 2-PDAs are more powerful than 1-PDAs.
b. Show that 3-PDAs are not more powerful than 2-PDAs.
(Hint: Simulate a Turing machine tape with two stacks.)

3.9 Solution:

a. It is easy to show by example: 1-PDA is not powerful enough to recognize $L = \{a^n b^n c^n |$ where $n \geq 0\}$, but we can construct 2-PDA that will recognize L as following:

   1. Put all a onto the first stack, and all b onto the second stack.
   2. When reading c pop both stack simultaneously. Accept if input and both stack are empty, reject otherwise.
   3. If encounter symbols out of order (b, c before a, or a between b and c) reject.

b. To show that 3-PDAs are not more powerful than 2-PDAs, it is enough to show that 2-PDA can simulate a regular TM.
   TM can be simulated by a 2-PDA as follows:
   First stack is responsible for handling everything on the left side of the TMs head, and right side for handling everything on the right side of the TMs head. On every M left transition S $\delta(q_i, c_i) = (q_j, c_j, L)$ PDA pops c_i off stack 2, pushes c_j into stack 2, pops stack 1 and pushes the character into stack 2, and goes from state q_i to q_j. Analogous for $\delta(q_i, c_i) = (q_j, c_j, R)$

   Regular Turing Machine is equal in computing capability to all other variations of computing machines and it can be simulated by 2-PDA, therefore 2-PDA and 3-PDA are equivalent.

Problem 3.11

A **Turing machine with doubly infinite tape** is similar to an ordinary Turing machine, but its tape is infinite to the left as well as to the right. The tape is initially filled with blanks except for the portion that contains the input. Computation is defined as usual except that the head never encounters an end to the tape as it moves leftward. Show that this type of Turing machine recognizes the class of Turing-recognizable languages.

Solution:

1. It easy to show that **doubly infinite tape TM** can simulate regular TM by not using the tape on the left side of the input.
2. **doubly infinite TM** can be easily simulated by **double tape TM**, by using second tape as the left side of the **doubly infinite TM**, and since **multi-tape TM** is equivalent to regular TM as it was proven in Sipser Theorem 3.13.

Problem 3.15
Show that the collection of decidable languages is closed under the operation of
Problem 3.15 (b)
concatenation.

Solution:
For any two decidable languages L_1 and L_2, let M_1 and M_2 be TMs that decide them. Concatenation of two languages L_1 and L_2 is language $C = \{xy|$ where $x \in L_1$ and $y \in L_2\}$. We construct TM M' that decides the concatenation of L_1 and L_2 that will non-deterministically split input w into two x and y, two check all possible combinations, i.e:

"On input w:

1. Non-deterministically split w into xy.
2. Run M_1 on x, if M_1 rejects reject, otherwise goto next.
3. Run M_2 on y, if M_2 accepts accept, otherwise reject.

Problem 3.15 (d)
complementation.

Solution:
For any decidable language L, let M be TM that decides it. We construct TM M' that decides the complement of L:

"On input w:

1. Run M on w. If it rejects accept, otherwise reject. "