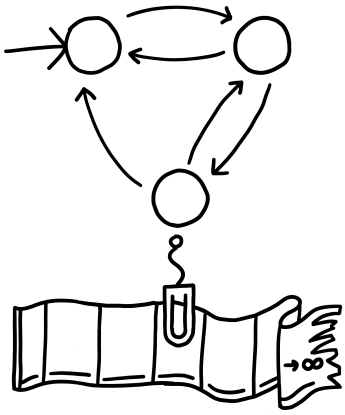
CS 4510 Automata and Complexity February 20th 2023

Lecture 10: Turing Machines

Lecturer: Abrahim Ladha *Scribe(s): Michael Wechsler*

Turing machines are so interesting, every lecture for the remainder of the course will be about Turing machines. I want this or that interesting sub-topic to be its own lecture, so that leaves us with nothing for today except definitions and programming. Recall the limitations of a PDA. You pop something out the stack, its gone into the ether, forever. What if we could iterate over our memory structure in a non destructive way. Our motivation is then to just give a DFA a tape.



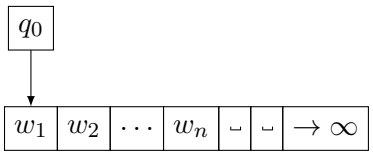
1 Definitions

A Turing machine is a tuple: $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where:

- Q : finite set of states
- Σ : finite input alphabet
- Γ : finite tape alphabet
  - $\textvisiblespace \in \Gamma$, this is the symbol for a blank space on the tape[1]
  - $\textvisiblespace \notin \Sigma$
  - $\Sigma \subsetneq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L,R}\}$ is our (deterministic) transition function.
- $q_0 \in Q$: denoted start state
- $q_a, q_r \in Q$: denoted accept and reject states

[1]For latex, you may use either `textvisiblespace` or `sqcup`

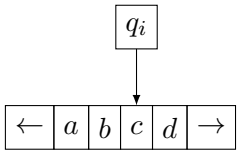
A Turing Machine is initialized with $w_1 w_2 ... w_n$ on the leftmost cells of the tape, and the tapehead on w_1. All other cells initialize to $\sqcup$.



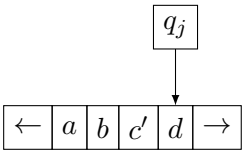
1.1 Configurations

A configuration of a Turing Machine is a string encoding of an instantaneous description, or snapshot, of a Turing Machine. We say a configuration C yields C' if C' follows C after one step of the transition function.. We write this as $C \vdash C'$. The entire description of the Turing machine at some moment is just the contents of the tape, the current state, and the position of the tape head. We encode these together as one string as follows.

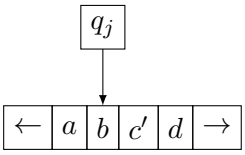
Consider the tape below



If $\delta(q_i, c) = (q_j, c', \mathrm{R})$, then $abq_icd \vdash abc'q_jd$



If $\delta(q_i, c) = (q_j, c', \mathrm{L})$, then $abq_icd \vdash aq_jbc'd$



$\star$ NOTE: $\vdash$ means yields

Notice that for sequential configurations, only a small local portion of the configuration is changed. The initial configuration of any Turing machine is always $q_0 w_1 \ldots w_n$. We may define the accepting and rejecting configurations appropriately, if they contain the accept or reject state. We say a Turing machine accepts $w_1 \ldots w_n$ if there exists a sequence of configurations

$C_0 = q_0 w_1 \ldots w_n$
$C_i \vdash C_{i+1}$
c_k is accepting

1.2 Create a Turing Machine, M, for $\{w\#w \mid w \in \Sigma^*\}$

Idea

mark checked as x	$abaa...\#abaa...$
skips xs to keep checking	$xxaa...\#xxaa...$

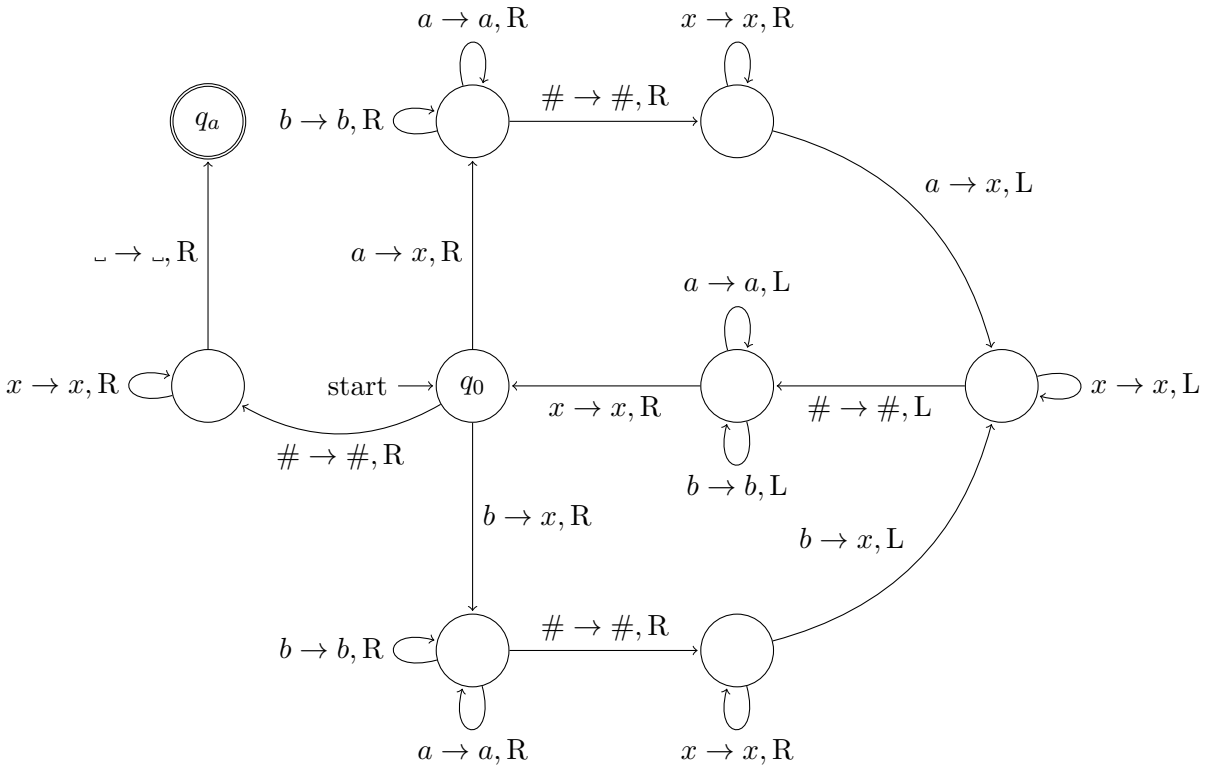
1.2.1 Pseudocode

M on input w:

1. mark symbol, keep track in states
2. loop right until #
3. loop past any x
4. if last marked = head:
    (a) mark
5. loop left until #
6. loop past any unmarked
7. reset to first unmarked (repeat from step 1)
8. if no symbols besides x before hash remain:
    (a) if no symbols besides x after hash remain:
        i. accept
9. reject

1.2.2 State Diagram

We can attempt to give a state diagram for this language. We will omit q_r, as it gets too messy. Undefined transitions in this diagram, you should understand to mean implicit rejection. Note how we have two branches, if we mark a or b. We use the states of the machine to keep track of what we have seen.

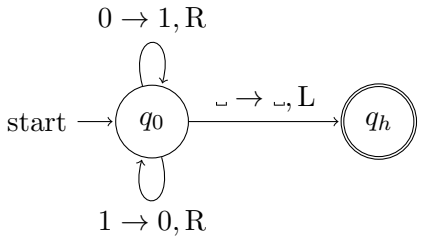


2 Computation

Unlike previous models, Turing machines do more than just decide languages. They can also compute! A function $f : \Sigma^* \to \Sigma^*$ is **Turing-Computable** (or just computable) if there exists a Turing Machine for all inputs w, which when initialized with w on the tape, halts with just $f(w)$ on its tape.

2.1 f : Bit Flips

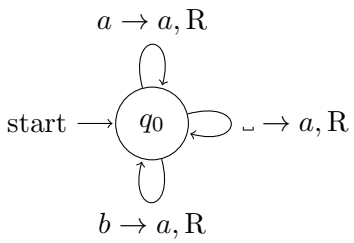
$\star$ NOTE: q_h is the halt state (accepting and rejecting do not matter here)



Example: Configurations for input 101

- q_0101
- $0q_001$
- $01q_01$
- $010q_0\sqcup$
- $01q_h0\sqcup$ HALTS in 4 steps

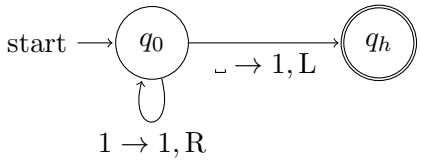
2.2 Turing Machines do NOT have to halt



The Turing Machine that writes a in every cell forever. Regardless of what it sees on its tape, its forever will march right, never stopping.

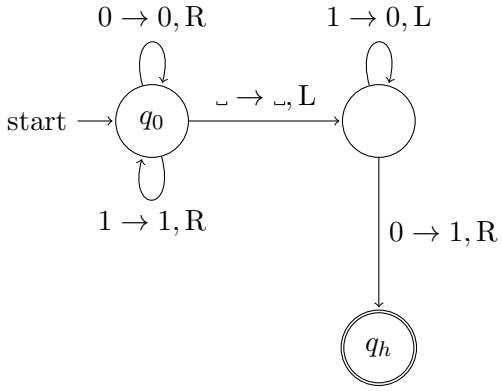
2.3 Successor Function: $S(x) = x + 1$

2.3.1 Unary



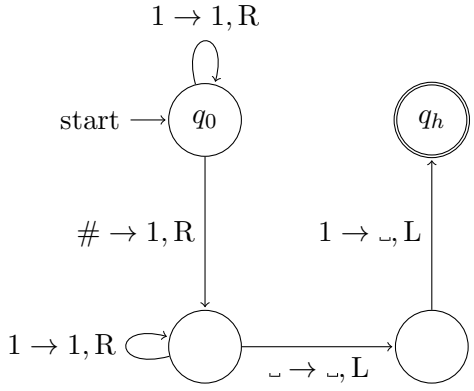
We begin with 1^x on the tape Halt with 1^{x+1}. The idea is we just get to loop to the end and toss a stick onto the pile.

2.3.2 Binary



For simplicity suppose the input is always given with a leading zero. We begin on the left on the input, so first we move all the way to the right. When adding 1 in binary, we loop from the right zeroing out all 1s until we find the first 0 and make it a 1.

2.4 Addition of 2 Numbers



$\mathrm{add}(x, y) = x + y$. Lets begin with $1^x \# 1^y$ on the tape. Halt with 1^{x+y} on the tape. The simplest idea is to replace the # with a 1 and remove the last 1 at the end. It would be challenging, but convince yourself you could give a Turing machine for addition in binary.

3 Decidability vs Recognizability

Recall, a function $f : \Sigma^* \to \Sigma^*$ is **Turing-Computable** (or computable) if there exists a Turing Machine for all inputs w, which when initialized with w on the tape, halts with $f(w)$ on its tape.

Additionally, a language, L, is **Turing-Decidable** (or just decidable or recursive) if there exists a Turing Machine, M, such that

- $w \in L \iff M$ accepts w
- $w \notin L \iff M$ rejects w

Notice that for every input, a decide always halts. A language, L, is **Turing-Recognizable** (or just recognizable or recursively-enumerable) if there exists a Turing Machine, M, such that

- $w \in L \iff M$ accepts w
- $w \notin L \iff M$ rejects w or gets stuck in a loop

We allow a recognizer to loop on some inputs. If the answer is supposed to be yes, it must always halt and accept. If the answer is suppose to be no, then it may halt and reject, or loop. It is clear to see that every decidable language is also recognizable, but is every recognizable language also decidable? We shall see.