Turing Machines

Mark Greenstreet, CpSc 421, Term 1, 2008/09

- A simple example
- Mathematical definition
- More examples

Background

- A DFA or NFA has a fixed set of states.
  - Thus, a DFA can only remember a bounded amount about its input no matter how long the input is.
  - We used this to show that there are languages that cannot be recognized by any DFA.
- A PDA has a finite controller and an unbounded stack.
  - The stack enables the PDA to store arbitrarily large amounts of data.
  - But, it can only access the top of stack:
    - To reach data that is further down, it must “pop” the intervening data items off the stack.
    - The finite controller can only remember a bounded amount about the stuff that has been popped of.
    - This leads to the limitations of PDAs – there are langauges that cannot be recognized by any PDA.

Turing Machines

- A Turing Machine has a (deterministic) finite state controller, and . . .
- a tape that it can read and write.
  - The tape is unbounded to the right.
  - The tape initially holds the input string.
  - The tape beyond the input string is initially filled with an infinite string of blanks, $\square$.
- the finite state controller has two special states:
  - q_{accept}: If the machine ever reaches this state, it halts and accepts the stringg.
  - q_{reject}: If the machine ever reaches this state, it halts and rejects the string.
  - If M is a Turing Machine, then the language recognized by M is written $L(M)$ and is the set of all strings for which the TM reaches the q_{accept} state.
- at each step:
  - M reads the symbol at its current position on the tape.
  - Based on that symbol and it current state, the machine:
    - Writes a symbol at the current position;
    - Transitions to a new state; and
    - Moves one square to the left or right.

Turing Machines (diagram)

My mouse isn't working right. You can draw it here.

All strings that contain three a's

- Let $\Sigma = \{$a, b$\}$.
- M has six states:
  - q_0 is the initial state: The machine has read 0 a's.
  - q_1, q_2 and q_3: the machine has read 1, 2 or 3 a's respectively.
  - q_{accept}: the machine reaches the end of the string after reading 3 a's.
  - q_{reject}: the machine has read more than 3 a's or reaches the end of the string having read fewer than 3 a's.
- The transitions:

current state	current tape symbol	next state	next tape symbol	move head
q_0	a	q_1	a	right
q_0	b	q_0	b	right
q_0	$\square$	q_{reject}	$\square$	right
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

All strings that contain three a's

- M has six states:
- The transitions:

current state	current tape symbol	next state	next tape symbol	move head
q_0	a	q_1	a	right
q_0	b	q_0	b	right
q_0	$\Box$	q_{reject}	$\Box$	right
q_1	a	q_2	a	right
q_1	b	q_1	b	right
q_1	$\Box$	q_{reject}	$\Box$	right
q_2	a	q_3	a	right
q_2	b	q_2	b	right
q_2	$\Box$	q_{reject}	$\Box$	right
q_3	a	q_{reject}	a	right
q_3	b	q_3	b	right
q_3	$\Box$	q_{accept}	$\Box$	right

All strings that contain three a's

You can draw the diagram here.

An Equivalent Program

```
state = q_0;
while(true) {
    switch(state) {
        case q_0:
            switch(currentSymbol) {
                case a:
                    write(a); state = q_1; move(right); break;
                case b:
                    write(b); state = q_0; move(right); break;
                case □:
                    write(□); state = q_reject; move(right); break;
            }
        case q_1: . . .
        case q_2: . . .
        case q_accept: accept();
        case q_reject: reject();
    }
}
```

$\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$: Strategy

- We can't count the number of a's, b's or c's with our finite control (you can't do it with Java int's long's either (why?)).
- We can zig-zag back and forth across the tape, matching up a's, b's and c's.
- Plan:
  - If the tape starts with an a, cross it off
    - scan to the right until we find a matching b, and cross it off.
    - continue scanning to the right until we find a matching c, and cross it off
    - Return to the beginning of the tape, and repeat the procedure.
  - We're done when. . .
    - We cross of every symbol – then accept ☺.
    - We fail to find a b or c when scanning to the right – reject ☹.
    - We still have some b's or c's left over after reading the last a – reject ☹.
  - Note:
    - When we return to the beginning of the a's, we need to be able to distinguiah having read all of the input from not having enough a's.
    - Solution: we'll use a different symbol for crossing off a's.

A program for $a^n b^n c^n$

```
while(true) {
    if(currentSymbol == □) accept();
    if(currentSymbol == a) {
        write(A); move(right)
        while(currentSymbol ∈ {a, B}) move(right);
        if(currentSymbol == b) {
            write(B); move(right);
        } else reject();
        while(currentSymbol ∈ {b, C}) move(right);
        if(currentSymbol == c) {
            write(C); move(left);
        } else reject();
        while(currentSymbol != A) move(left);
        move(right);
    } else if(currentSymbol ∈ {B, C}) move(right);
    else reject();
}
```

Compiling to a Turing Machine

q_0:	while(true) {
q_0:	if(currentSymbol == $\square$) {
q_{accept}:	accept();
q_0:	} else if(currentSymbol == a) {
q_0:	write(A); move(right)
q_1:	while(currentSymbol $\in$ {a, B}) move(right);
q_1:	if(currentSymbol == b) {
q_1:	write(B); move(right);
q_1:	} else
q_{reject}:	reject();
q_2:	while(currentSymbol $\in$ {b, C}) move(right);
q_2:	if(currentSymbol == c) {
q_2:	write(C); move(left);
q_2:	} else
q_{reject}:	reject();
q_3:	while(currentSymbol != A) move(left);
q_3:	move(right);
q_0:	} else if(currentSymbol $\in$ {B, C}) {
q_0:	move(right);
q_0:	} else reject();
	}

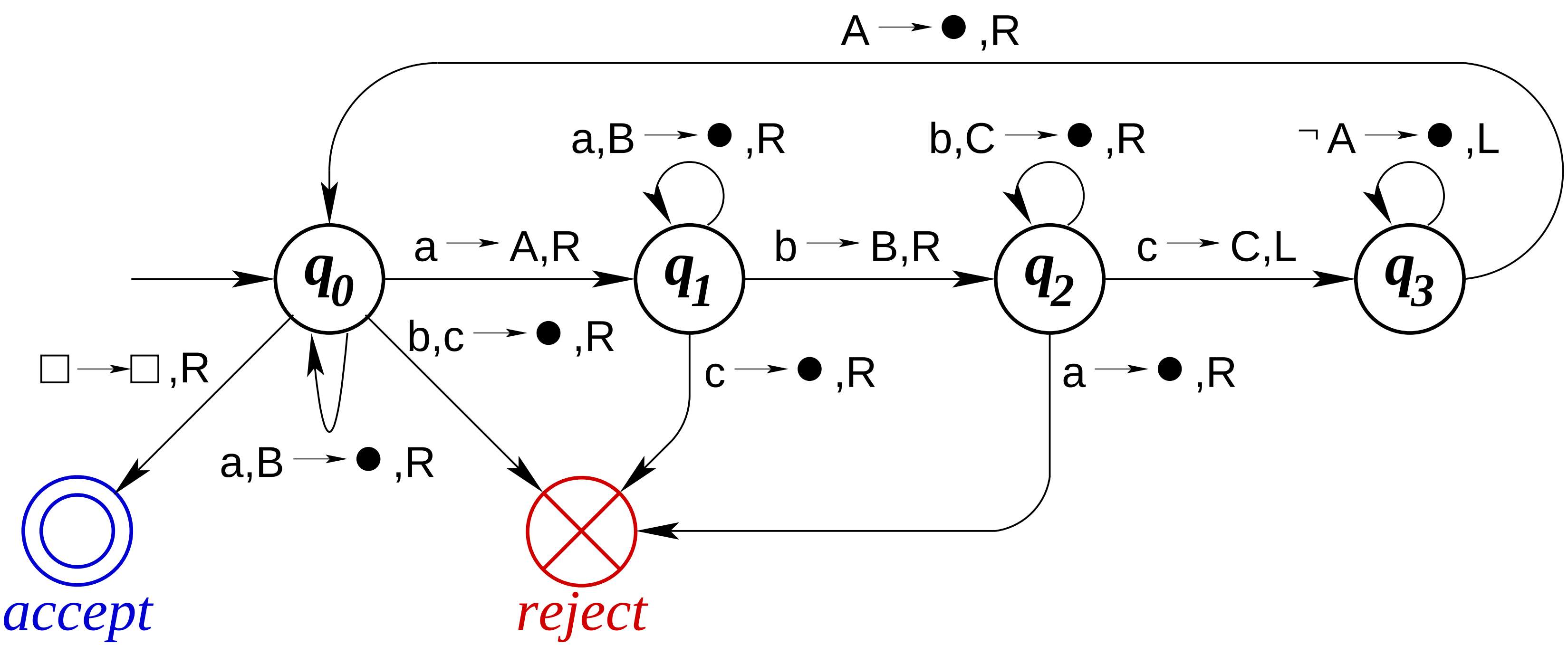
A Turing Machine for $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$

- Input alphabet: $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$.
- States: $Q = \{q_0, q_1, q_2, q_3, q_4, q_{accept}, q_{reject}\}$.
- Tape alphabet: $\Gamma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{A}, \mathsf{B}, \mathsf{C}, \Box\}$.
- Transitions:

q_0 :	$(q_0, \mathsf{a}) \longrightarrow (q_1, \mathsf{A}, \text{right})$	$(q_0, \Box) \longrightarrow (q_{accept}, \Box, \text{right})$
	$(q_0, \{\mathsf{B}, \mathsf{C}\}) \longrightarrow (q_0, \bullet, \text{right})$	$(q_0, other) \longrightarrow (q_{reject}, \bullet, \text{right})$
q_1 :	$(q_1, \{\mathsf{a}, \mathsf{B}\}) \longrightarrow (q_1, \bullet, \text{right})$	$(q_1, \mathsf{b}) \longrightarrow (q_2, \mathsf{B}, \text{right})$
	$(q_1, other) \longrightarrow (q_{reject}, \bullet, \text{right})$	
q_2 :	$(q_2, \{\mathsf{b}, \mathsf{C}\}) \longrightarrow (q_2, \bullet, \text{right})$	$(q_2, \mathsf{c}) \longrightarrow (q_3, \mathsf{C}, \text{left})$
	$(q_2, other) \longrightarrow (q_{reject}, \bullet, \text{right})$	
q_3 :	$(q_3, \Gamma - \{\mathsf{A}\} \longrightarrow (q_3, \bullet, \text{left})$	$(q_3, \mathsf{A}) \longrightarrow (q_0, \bullet, \text{right})$

- Writing a $\bullet$ on the tape means writing the same symbol that was read.

A Turing Machine for $\mathbf{a}^n\mathbf{b}^n\mathbf{c}^n$



To avoid clutter, I've omitted edges for transitions that can never occur. These are labeled "$other$" in the table on the previous slide.

An Accepting Run

step	state	tape
0	q_0	aaabbbccc$\square^*$
1	q_1	Aaabbbccc$\square^*$
2	q_1	Aaabbbccc$\square^*$
3	q_1	Aaabbbccc$\square^*$
4	q_2	AaaBbbccc$\square^*$
5	q_2	AaaBbbccc$\square^*$
6	q_2	AaaBbbccc$\square^*$
7	q_3	AaaBbbCcc$\square^*$
8	q_3	AaaBbbCcc$\square^*$
9	q_3	AaaBbbCcc$\square^*$
10	q_3	AaaBbbCcc$\square^*$
11	q_3	AaaBbbCcc$\square^*$
12	q_3	AaaBbbCcc$\square^*$
13	q_0	AaaBbbCcc$\square^*$

The purple symbol indicates the current tape head position.

An Accepting Run

step	state	tape
13	q_0	AaaBbbCcc□*
14	q_1	AAaBbbCcc□*
15	q_1	AAaBbbCcc□*
16	q_1	AAaBbbCcc□*
17	q_2	AAaBBbCcc□*
18	q_2	AAaBBbCcc□*
19	q_2	AAaBBbCcc□*
20	q_3	AAaBBbCCc□*
21	q_3	AAaBBbCCc□*
22	q_3	AAaBBbCCc□*
23	q_3	AAaBBbCCc□*
24	q_3	AAaBBbCCc□*
25	q_3	AAaBBbCCc□*
26	q_0	AAaBBbCCc□*

An Accepting Run

step	state	tape
26	q_0	AAaBBbCCc□*
27	q_1	AAABBbCCc□*
28	q_1	AAABBbCCc□*
29	q_1	AAABBbCCc□*
30	q_2	AAABBBCCc□*
31	q_2	AAABBBCCc□*
32	q_2	AAABBBCCc□*
33	q_3	AAABBBCCC□*
34	q_3	AAABBBCCC□*
35	q_3	AAABBBCCC□*
36	q_3	AAABBBCCC□*
37	q_3	AAABBBCCC□*
38	q_3	AAABBBCCC□*
39	q_0	AAABBBCCC□*

An Accepting Run

step	state	tape
39	q_0	AAABBBCCC□*
40	q_0	AAABBBCCC□*
41	q_0	AAABBBCCC□*
42	q_0	AAABBBCCC□*
43	q_0	AAABBBCCC□*
44	q_0	AAABBBCCC□*
45	q_0	AAABBBCCC□ □*
47	q_{accept}	AAABBBCCC□ □□*

Formal Definition of Turing Machines

- A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where
  - Q is a finite set, the states.
  - Σ is a finite set, the input alphabet.
  - $\Gamma \supset \Sigma$ is a finite set, the tape alphabet.
  - $\delta : (Q \times \Gamma) \rightarrow (Q \times \Gamma \times \{L, R\})$ is the transition function.
  - $q_0 \in Q$ is the initial state.
  - $q_{accept} \in Q$ is the accepting state.
  - $q_{reject} \in Q$ is the rejecting state.

Turing Machine Configurations

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be a Turing machine.
- A configuration consists of
  - A state, q, the current state of the Turing Machine.
  - A string w, the tape currently holds $w\Box^*$.
  - A position: where the read/write head is along the tape.
- We write uqv where $u \in \Gamma^*$ and $v \in \Gamma^*$ to indicate that a Turing machine in in a configuration where
  - The controller is in state q.
  - The tape contents are $uv\Box^*$.
  - The read/write head is positioned at the first symbol of v.

Turing Machine Moves

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be a Turing machine.
- Let q be a state in $Q - \{q_{accept}, q_{reject}\}$. M can move from configuration $uqcv$ to configuration $u'q'v'$ for some $u, v, u', v' \in \Gamma^*$, $q, q' \in Q$, and $c \in \Gamma$, iff
  - There is some d such that $\delta(q, c) = (q', d, R)$, and
    - $v \neq \epsilon$ and $u' = ud$, and $v' = v$; or
    - $v = \epsilon$ and $u' = ud$, and $v' = \square$; or
  - There is some d such that $\delta(q, c) = (q', d, L)$, and
    - $u = u'b$ and $v' = bdv$; or
    - $u = u' = \epsilon$ and $v' = dv$.
- If C_1 and C_2 are configurations and M can move from C_1 to C_2, then we write $C_1 \xrightarrow{M} C_2$. If M is obvious from context, we write $C_1 \rightarrow C_2$.

Turing Machine Moves

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be a Turing machine.
- Let q be a state in $Q - \{q_{accept}, q_{reject}\}$. M can move from configuration $uqcv$ to configuration $u'q'v'$ for some $u, v, u', v' \in \Gamma^*$, $q, q' \in Q$, and $c \in \Gamma$, iff ...
- If C_1 and C_2 are configurations and M can move from C_1 to C_2, then we write $C_1 \xrightarrow{M} C_2$. If M is obvious from context, we write $C_1 \rightarrow C_2$.
- If $C = uqv$ is a configuration with $q = q_{accept}$, we say that C is an accepting configuration.
  Likewise if $q = q_{reject}$, we say that C is a rejecting configuration.
- Accepting and rejecting configuration are halting configurations: the Turing machine makes no further moves from such a configuration.

Turing Machine Acceptance

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be a Turing machine.
- M accepts input w iff there is a set of configurations C_0, C_1, $\ldots C_i$ such that
  - $C_0 = q_0 w$;
  - For all j in $0 \ldots i-1$, $C_j \stackrel{M}{\rightarrow} C_{j-1}$;
  - C_i is an accepting configuration.
- M rejects w iff there is a set of configurations that ends in a rejecting configuration.
- M loops on input w if M neither accepts nor rejects w. This means that w executes forever on input w.

Languages recognized by TMs

- Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be a Turing machine.
- M recognizes language A iff
  - M accepts w iff $w \in A$.
  - If $w \notin A$, then M may either reject or loop on input w.
- M decides language A iff
  - If $w \in A$ then M accepts w; and
  - if $w \notin A$ then M rejects w.
  - (In other words, M never loops.)

Turing Languages

- A language is Turing recognizable iff there is some Turing machine that recognizes it (such a Turing machine may loop).
- A language is Turing decidable iff there is some Turing machine that decides it (i.e. no looping).
- Every Turing decidable language is Turing recognizable, but
- We will show
  - there are Turing recognizable languages that are not Turing decidable (next week)
  - there are lanuages that not even Turing recognizable (later).

This coming week

- Reading
  - October 17 (today): *Sipser* 3.1.
  - October 20 (Monday): *Sipser* 3.2.
  - October 22 (Wednesday): *Sipser* 3.3.
  - October 24 (a week from today): *Sipser* 4.1.
- Homework
  - October 17 (today): Homework 4 due; homework 6 goes out.
  - October 20 (Monday): Homework 5 due.
  - October 24 (a week from today): Homework 6 due; homework 7 goes out.