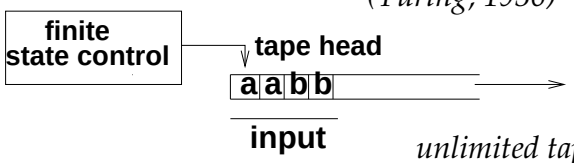
Turing Machines

Most powerful computational model

(Turing, 1936)



unlimited tape

Operations:

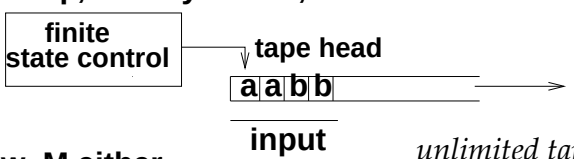
Read symbol, write symbol at head
Move Left (L) or Right (R)

Initial configuration: input starting at left end, with blanks following; tape head at left end

Will let us study the power and limits of computability

Computation on a Turing Machine

Given current state q, current symbol a $\xrightarrow{\delta}$
new state q', new symbol z, move head L or R



unlimited tape

On input w, M either

1. **Enters the** *accept state* q_{acc} **and accepts w, or**
2. **Enters the** *reject state* q_{rej} **and rejects w, or**

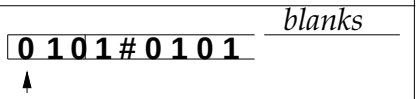
(In these 2 cases, we say M halts on input w)

3. **Does neither 1 or 2, in which case it rejects w**

(In case 3, we say M does not halt on input w)

Turing Machine Example

L = {w#w | w in {0,1}* }



1. Scan the input to ensure its in {0,1}*#{0,1}* If not, reject.
2. Zig-zag across tape to corresponding positions on either side of #. If the symbols are the same, check them off. If not, reject.
3. When all the symbols to the left of # are checked off, check on the right of # to make sure there are no unchecked symbols. If none, accept. If some, then reject.

Formal Definition of TM

A *TM M* is a 7-tuple (Q, Σ, Γ, δ , q_o, q_a, q_r)

1. Q is a finite set of *states*
2. Σ is the *input alphabet*, not containing $\sqcup$
3. Γ is the *tape alphabet*, with $\Sigma \subseteq \Gamma$, $\sqcup$ in Γ
4. δ : Q x $\Gamma \rightarrow$ Q xΓ x {L,R}
5. q_o in Q, the *start state*
6. q_a in Q, *accept state*
7. q_r in Q, *reject state* $q_r \neq q_a$

Qu: Deterministic or Nondeterministic?

Configurations of TM M

A *configuration* is a string uqv where
q is a state in Q
uv is the current (nonblank) tape contents
M's head is reading the first symbol of v

The *start configuration* is $q_0 w$ (on input w)

If M ever tries to move off the left end of tape, the head stays in the same place:
qcv $\longrightarrow$ q'bv *(Even though move is to L)*

An *accepting configuration* is one with q_a

A *rejecting configuration* is one with q_r
(Head can be anywhere)

A *halting configuration* is an accepting or rejecting configuration.

Acceptance of TM M

A TM M *accepts* input w if there is a sequence of configurations C_1, ... C_k with

1. C_1 the start configuration
2. C_i yields C_{i+1} by following δ one step
3. C_k is accepting

L(M) = {w | M accepts w}

A is *Turing-recognizable* if A = L(M) for some TM M.

If M always halts, it is a *decider.*

A is *Turing-decidable* if A = L(M) for some decider TM M.
(We say M decides A)

Example TM

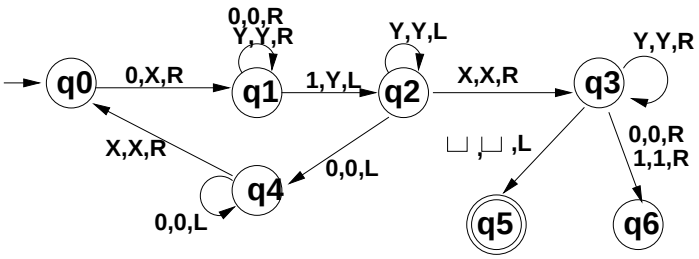
$L = \{ 0^n 1^n \mid n \geq 1 \}$

Q = {q0, q1, q2, q3, q4, q5, q6}

Σ = {0,1} Γ = {0,1, X, Y, ␣}

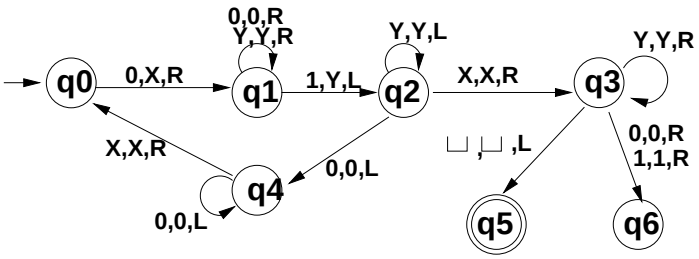
q0 start q5 accept q6 reject

δ given by



Example TM

Configurations on input 0011 010



Example TM

$L = \{ a^i b^j c^{i+j} \mid i,j \geq 1 \}$

Describe TM for L:

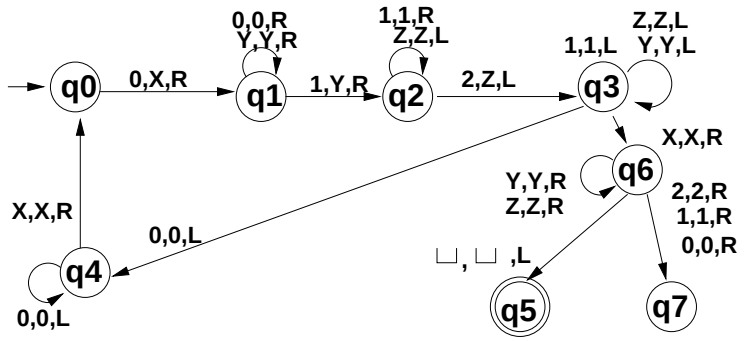
1. Scan the input to make sure its in a*b*c*. Reject if not.
2. Zig-zag between a's and c's. Replace each leftmost a with an X, and then the leftmost c with a Z. Do this until run out of a's. If run out of c's, reject.
3. Zig-zag between b's and c's. Replace each leftmost b with a Y, and then the leftmost c with a Z. Do this until run out of b's. If run out of c's, reject.
4. When run out of b's, move right; if any c's left, then reject; otherwise, accept.

Example TM

$L = \{ 0^n 1^n 2^n \mid n > 1 \}$

$\Sigma = \{0,1,2\}$ $\Gamma = \{0,1,2\ X, Y, Z, \sqcup\}$

Note: can't accept with any PDA!



TM Construction

Methods for programming TM's

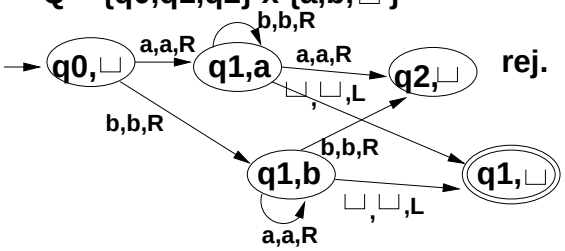
1. Storage in states

Old states Q New states Q x I1 x ... x Ik

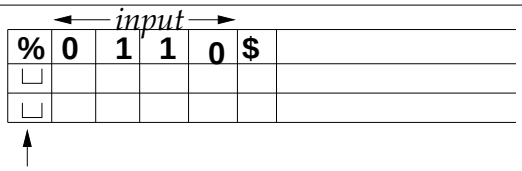
Ex. L given by ab* ∪ ba*

Construct TM M that stores first symbol, then makes sure not in rest of input.

Q = {q0,q1,q2} x {a,b,␣}



Multiple tracks on single tape



Each symbol is a k-tuple (here, k = 3)

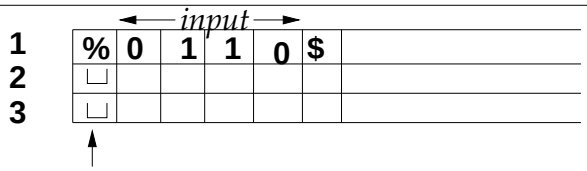
Ex. **Construct M to accept if input is a binary representation of a prime number n**

(n is prime if has no other factors exept n and 1 that divide it evenly)

Procedure: Check to see if 2, 3, ... n-1 divide n; if any do, reject. if none, accept.

How use tracks?

Multiple tracks on single tape



On track 2, cycle through numbers m, starting with 2 and ending with n-1.
For each such m, use track 3 to see if n is divisible by m: start by copying track 1 to track 3. Then repeatedly subtract m on track 2 from track 3, until you get a remainder on track 3 which is less than m. If remainder = 0, then m divides n, and so M rejects. If remainder is not 0, add 1 to m to obtain the new m, and continue, till get to n.

How do subtraction? test if one number less than another?