Chapter 3
The Church-Turing Thesis

Overview

- so far, we have presented several models of computing devices
  - finite automata for devices with a small amount of memory
  - pushdown automata for devices with unlimited memory usable in a LIFO stack
  - other simple tasks are beyond the capabilities of these models
    - too restricted to serve as models of general-purpose computers

Turing Machines

- we now turn to more powerful model
  - Turing machine
  - proposed by Alan Turing in 1936

- Turing machine
  - similar to FA
  - but with unlimited and unrestricted memory
  - more accurate model of a general-purpose computer
    - can do everything a real computer can do
    - still certain problems that it cannot solve
      - these problems may be beyond the theoretical limits of computation

Turing Machines

- Turing machine
  - infinite tape as unlimited memory
  - tape head can read and write symbols and move around on the tape
  - initially, tape contains input string
    - blank everywhere else
  - to store information, machine can write on tape
  - to read information, tape head can move back over it
  - machine continues computing until it produces an output
    - accept and reject by entering accept or reject states
    - otherwise, it will go on forever, never halting

Turing Machines

- differences between FA and Turing machines
  - Turing machines can both write and read on the tape
  - the read-write head can both move left and right
  - the tape is infinite
  - states for rejecting and accepting take effect immediately

Turing Machines

- example: Turing machine M_1 for testing membership in language $B = \{w\#w \mid w \in \{0,1\}^*\}$
  - accept if input in B; reject otherwise
  - as before, put yourself in place of the Turing machine
    - imagine standing on input with millions of characters
    - goal is to determine if input is in B
      - i.e., two identical strings separated by #
    - string too long to remember, but you can move back and forth on input and mark on it
    - strategy: zig-zag on corresponding places of two sides of # and determine if they match
      - mark tape to keep track of correspondences

Turing Machines

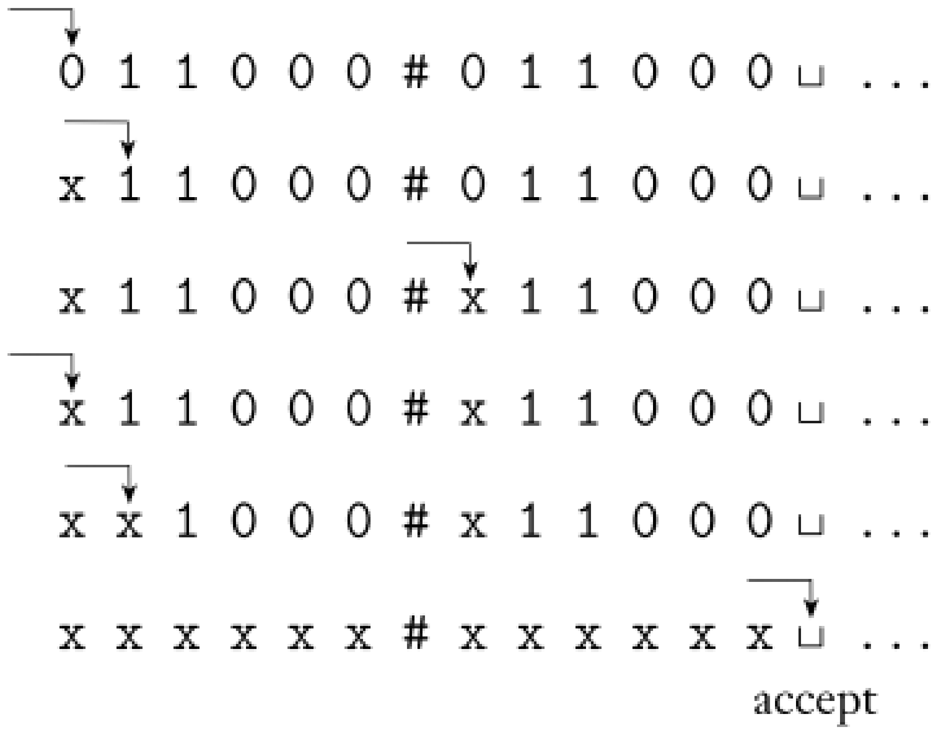
- example: Turing machine M_1 for testing membership in language $B = \{w\#w \mid w \in \{0,1\}^*\}$ (cont.)
  - will work this way
    - M_1 makes multiple passes over input
    - on each pass, matches characters on each side of #
    - crosses off each symbol as it is examined
    - if all symbols crossed off → match
      - goes into accept state
    - mismatch
      - goes into reject state

Turing Machines

- example: Turing machine M_1 for testing membership in language $B = \{w\#w \mid w \in \{0,1\}^*\}$ (cont.)
  - algorithm
    - zig-zag across tape to corresponding positions on either side of #
      - check whether these positions contain the same symbol
        - if they do not, or no # is found → reject
      - cross off symbols as they are checked to keep track of which symbols correspond
    - when all symbols to the left of the # have been crossed off, check for remaining symbols on the right
      - if any symbols remain → reject
      - otherwise → accept

Turing Machines

- example: Turing machine M_1 for testing membership in language $B = \{w\#w \mid w \in \{0,1\}^*\}$ (cont.)
  - nonconsecutive snapshots of tape with input 011000#011000



Turing Machines

- previous example leaves out some details
- formal definition is a 7-tuple
  - transition function δ tells us how machine gets from one step to the next
    - $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$
    - when machine is in a certain state q and the head is over a tape square containing symbol a
    - if $\delta(q,a) = (r,b,L)$, writes symbol b replacing a, goes to state r, move Left

Turing Machines

A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets 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. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the 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 $q_{reject} \neq q_{accept}$.

Turing Machines

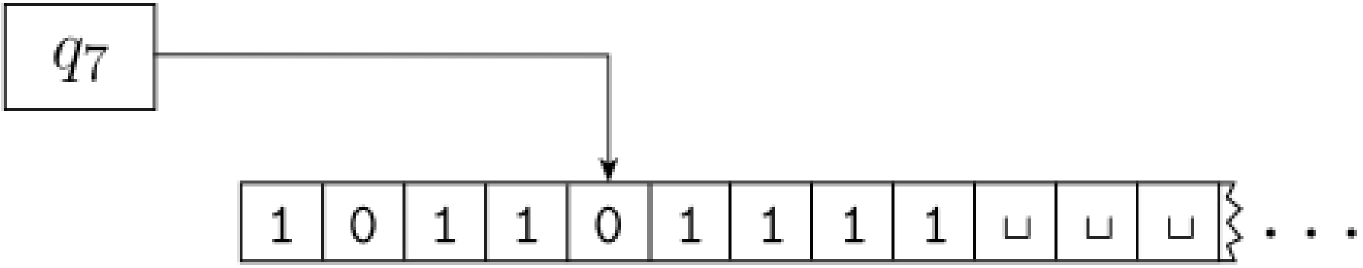
- a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ computes as follows
  - input $w = w_1 w_2 \ldots w_n \in \Sigma^*$ on leftmost n squares of tape
  - the rest of the tape is all blank symbols
  - head starts at leftmost square
  - Σ does not contain blank, so first blank appearing on tape marks the end of the input
  - M goes from state to state according to the rules of δ
    - if M tries to move its head left off left-hand end of tape, the head stays in the same place for that move
  - computation continues until either accept or reject state is entered
    - if neither occurs, M goes on forever

Turing Machines

- as a Turing machine computes, changes occur in the
  - current state
  - current tape contents
  - current head location
- a setting of these three items is called a configuration of the Turing machine
  - for state q and two strings u and v over the tape alphabet Γ
    - u q v is the configuration where the current state is q, tape contents is uv, and head location is at first symbol of v
    - only blanks after last symbol of v

Turing Machines

- example: $1011q_7 01111$
  - tape is 101101111
  - current state is q_7
  - head is currently on second 0



Turing Machines

- configuration C_1 yields C_2 if the TM can go from C_1 to C_2 in a single step

- formal description
  - suppose we have $a,b,c \in \Gamma$, $u,v \in \Gamma^*$, and states q_i and q_j

  - ua q_i bv and u q_j acv are two configurations (moving L)
    - therefore, ua q_i bv yields u q_j acv if $\delta(q_i,b) = (q_j,c,L)$

  - ua q_i bv and uac q_j v are two configurations (moving R)
    - therefore, ua q_i bv yields uac q_j v if $\delta(q_i,b) = (q_j,c,R)$

Turing Machines

- special cases occur when the head is at one of the ends of the configuration

  - for left-hand end, the configuration q_i bv yields q_j cv if moving to left
    - we're preventing the machine from going off left-hand end of tape

  - for right-hand end, the configuration ua q_i is equivalent to ua q_i _ since blanks follow the right-most character

Turing Machines

- start configuration is q_0w
  - machine is in start state q_0
  - head at leftmost position on the tape

- accepting configuration is q_{accept}
- rejecting configuration is q_{reject}

- accepting and rejecting configurations are halting configurations
  - do not yield further configurations

Turing Machines

- a Turing machine accepts input w if a sequence of configurations C_1, C_2,…,C_k exists where
  - C_1 is the start configuration M on input w
  - each C_i yields C_{i+1}
  - C_k is the accepting configuration
- the collection of strings M accepts is the language of M
  - or the language recognized by M, or L(M)
- a language is Turing-recognizable if some Turing machine recognizes it

Turing Machines

- when we start a Turing machine on an input, three outcomes are possible
  - accept
  - reject
  - loop (does not halt)

- a TM can fail to accept an input by entering the q_{reject} state and rejecting, or by looping
  - sometimes difficult to distinguish machine looping vs. just taking a long time

- prefer TMs that halt on all inputs (never loop)
  - such machines are called deciders
  - always make a decision to accept or reject

Turing Machines

- a decider that recognizes some language is also said to decide that language
  - call a language Turing-decidable, or simply decidable, if some Turing machine decides it

Examples of Turing Machines

- we could formally describe a TM with its 7-tuple
  - however, lots of information
  - alternatively, we will only give higher level descriptions
    - really just shorthand for formal counterpart

Examples of Turing Machines

- example: describe a TM M_2 that describes $A = \{0^{2^n} \mid n \geq 0\}$ (or the language consisting of all strings of 0s whose length is a power of 2)

M_2 = "on input string w:

1. sweep left to right across the tape, crossing off every other 0
2. if in stage 1, the tape contained a single 0 → accept
3. if in stage 1, the tape contained more than a single 0 and the number of 0s was odd → reject
4. return the head to the left-hand end of the tape
5. go to stage 1."

Examples of Turing Machines

- example: describe a TM M_2 that describes $A = \{0^{2^n} \mid n \geq 0\}$ (or the language consisting of all strings of 0s whose length is a power of 2), cont.

  - each iteration of stage 1 cuts the number of 0s in half
  - machine keeps track of whether the number of 0s seen is even or odd
    - if odd and > 1, original number of 0s is not a power of 2
      - input is rejected
    - if number of 0s seen is 1, original number of 0s must be a power of 2
      - input is accepted

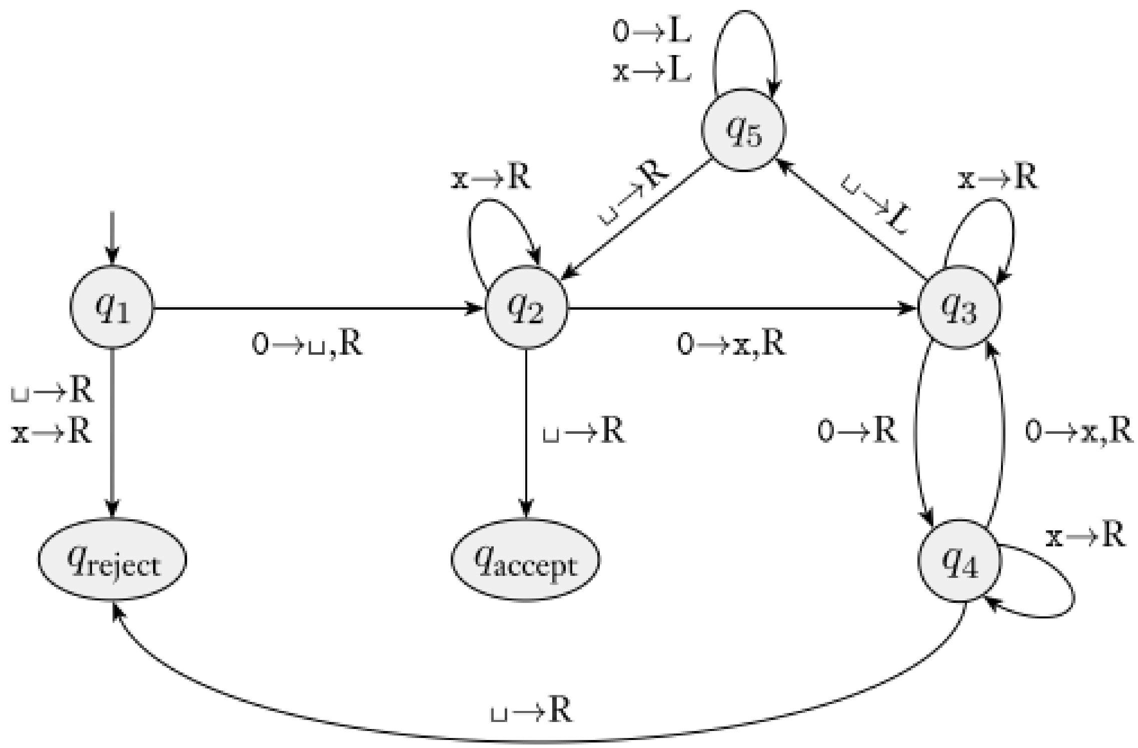
Examples of Turing Machines

- example: describe a TM M_2 that describes $A = \{0^{2^n} \mid n \geq 0\}$ (or the language consisting of all strings of 0s whose length is a power of 2), cont.

  - formal description of $M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$
    - $Q = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject})$
    - $\Sigma = \{0\}$
    - $\Gamma = \{0, x, \_\}$
    - δ described with state diagram (next slide)
    - start state = q_1
    - accept state = q_{accept}
    - reject state = q_{reject}

Examples of Turing Machines

- example: describe a TM M_2 that describes $A = \{0^{2^n} \mid n \geq 0\}$ (or the language consisting of all strings of 0s whose length is a power of 2), cont.
  - state diagram



Examples of Turing Machines

- example: describe a TM M_2 that describes $A = \{0^{2^n} \mid n \geq 0\}$ (or the language consisting of all strings of 0s whose length is a power of 2), cont.
  - state diagram explanation
    - $0 \rightarrow$ _,R appears on arc from q_1 to q_2
      - when in state q_1 with the head reading 0, go to q_2, write _, and move the head right
        - $\delta(q_1, 0) = (q_2, \_, R)$
    - machine begins by writing a blank over leftmost 0 on tape so that it can find the left-hand end in stage 4
      - could have used another symbol (like #), but we like to keep the tape alphabet small

Examples of Turing Machines

- example: describe a TM M_2 that describes $A = \{0^{2^n} \mid n \geq 0\}$ (or the language consisting of all strings of 0s whose length is a power of 2), cont.
  - sample run on input 0000

$q_1 0000$	$\sqcup q_5 \mathrm{x0x}\sqcup$	$\sqcup \mathrm{x} q_5 \mathrm{xx}\sqcup$
$\sqcup q_2 000$	$q_5 \sqcup \mathrm{x0x}\sqcup$	$\sqcup q_5 \mathrm{xxx}\sqcup$
$\sqcup \mathrm{x} q_3 00$	$\sqcup q_2 \mathrm{x0x}\sqcup$	$q_5 \sqcup \mathrm{xxx}\sqcup$
$\sqcup \mathrm{x0} q_4 0$	$\sqcup \mathrm{x} q_2 \mathrm{0x}\sqcup$	$\sqcup q_2 \mathrm{xxx}\sqcup$
$\sqcup \mathrm{x0x} q_3 \sqcup$	$\sqcup \mathrm{xx} q_3 \mathrm{x}\sqcup$	$\sqcup \mathrm{x} q_2 \mathrm{xx}\sqcup$
$\sqcup \mathrm{x0} q_5 \mathrm{x}\sqcup$	$\sqcup \mathrm{xxx} q_3 \sqcup$	$\sqcup \mathrm{xx} q_2 \mathrm{x}\sqcup$
$\sqcup \mathrm{x} q_5 \mathrm{0x}\sqcup$	$\sqcup \mathrm{xx} q_5 \mathrm{x}\sqcup$	$\sqcup \mathrm{xxx} q_2 \sqcup$
		$\sqcup \mathrm{xxx}\sqcup q_{\mathrm{accept}}$

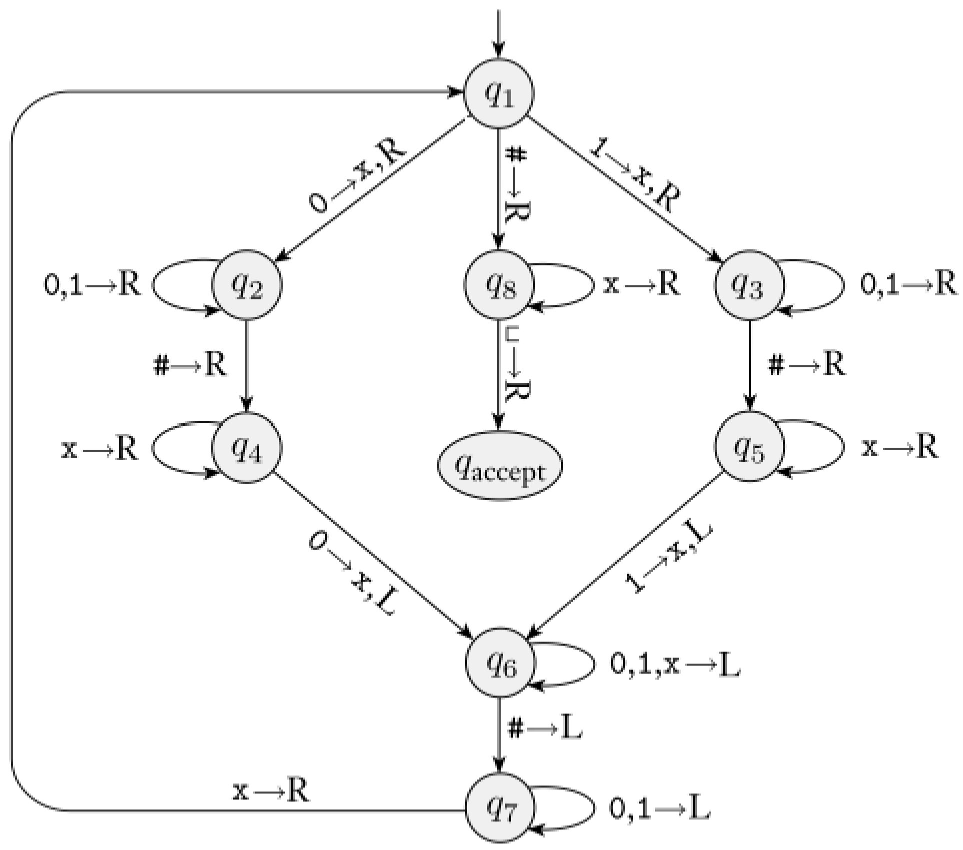
Examples of Turing Machines

- example: describe a Turing machine M_1 that describes $B = \{w\#w \mid w \in \{0,1\}^*\}$

  - $M_1 = (Q, \Sigma, \Gamma, \delta, q_1, q_{accept}, q_{reject})$ where
    - $Q = \{q_1, \ldots, q_8, q_{accept}, q_{reject})$
    - $\Sigma = \{0, 1, \#\}$
    - $\Gamma = \{0, 1, \#, x, \_\}$
    - δ described with state diagram (next slide)
    - start state = q_1
    - accept state = q_{accept}
    - reject state = q_{reject}

Examples of Turing Machines

- example: describe a Turing machine M_1 that describes $B = \{w\#w \mid w \in \{0,1\}^*\}$, cont.
  - state diagram



Examples of Turing Machines

- example: describe a Turing machine M_1 that describes $B = \{w\#w \mid w \in \{0,1\}^*\}$, cont.
  - state diagram explanation
    - label $0,1 \rightarrow R$ on arc from q_3 to itself
      - stay in q_3 and move to right when reading a 0 or 1 in state q_3
      - do not change symbol on tape
    - stage 1 implemented by q_1 through q_7
    - stage 2 by the remaining states
    - for clarity, reject state not shown
      - implicit rejection when no arc leaves state on symbol
        - e.g., no arc for # from q_5
        - in all such cases, head moves right on arc to q_{reject}

Examples of Turing Machines

- example: design a Turing machine M_3 that performs elementary arithmetic: $C = \{a^i b^j c^k \mid i \times j = k \text{ and } i,j,k \geq 1\}$

M_3 = "on input string w:

1. scan input from left to right to determine if it is a member of $a^+b^+c^+$ and reject if it isn't
2. return head to left-hand end of tape
3. cross off an a and scan right until b occurs
   - shuttle between b's and c's, crossing off one of each until all b's are gone
   - if all c's crossed off, but some b's remain → reject
4. restore crossed off b's
   - repeat stage 3 if another a to cross off
   - if all a's crossed off and all c's crossed off → accept
   - otherwise → reject."

Examples of Turing Machines

- example: design a Turing machine M_3 that performs elementary arithmetic: $C = \{a^i b^j c^k \mid i \times j = k \text{ and } i,j,k \geq 1\}$, cont.
  - closer look at stages of M_3
    - in stage 1, machine operates like a FA
      - no writing necessary as head moves left to right
      - keeps track by using states to determine if input in proper form
    - in stage 2, how does machine find left side of input?
      - finding right end is easy since it is terminated with blank
      - no terminating symbol on left end
      - can mark the left end with a blank when machine starts
      - alternatively, recall that if the machine tries to move left beyond the left end of the tape, it stays in the same place
        - to make a left end detector, we can write a special symbol at the current position while recording the symbol it replaced
        - it can then try to move left
        - if it is still over the special symbol, it must be on the left end
        - otherwise, there are other symbols, and the original symbol is restored
    - stages 3 and 4 are straightforward

Examples of Turing Machines

- example: describe a Turing machine M_4 that solves the element distinctive problem: given a list of strings over {0,1} separated by #s, accept if all strings are different

$$E = \{\#x_1\#x_2\#\ldots\#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

  - works by comparing x_1 with x_2 through x_l
  - then comparing x_2 with x_3 through x_l
  - etc.

Examples of Turing Machines

- example: describe a Turing machine M_4 that solves the element distinctive problem: given a list of strings over {0,1} separated by #s, accept if all strings are different, cont.

  $E = \{\#x_1\#x_2\#...\#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$

  M_4 = "on input w:
  1. place mark on top of leftmost symbol
     - if symbol was a blank → accept
     - if symbol was #, continue to next stage
     - otherwise → reject
  2. scan right to next # and place second mark on it
     - if no # encountered before blank, only x_1 was present → accept
  3. zig-zag and compare two strings to right of marked #s
     - if they are equal → reject
  4. move rightmost of two marks to next # symbol to the right
     - if no # encountered before blank, move leftmost mark to next # to its right and rightmost mark to the # after that (reset first string)
     - if no # is available for rightmost mark, all strings have been compared → accept
  5. go to stage 3."

Examples of Turing Machines

- example: describe a Turing machine M_4 that solves the element distinctive problem: given a list of strings over {0,1} separated by #s, accept if all strings are different, cont.

$$E = \{\#x_1\#x_2\#\ldots\#x_l \mid \text{each } x_i \in \{0,1\}^* \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$$

  - this machine illustrates the technique of marking tape symbols
    - in stage 2, the machine places a mark above the symbol #
    - in the implementation, there are two different symbols: # and $\dot{\#}$
  - whenever the machine makes a mark above a symbol, it means the second one
  - removing the mark means the machine writes the first one
  - used in a variety of situations – just include both in the alphabet

Examples of Turing Machines

- from the previous examples, the languages A, B, C, and E are decidable
- all decidable languages are Turing-recognizable, so these languages are also Turing-recognizable

Variants of Turing Machines

- different versions of Turing machines abound
  - multitape Turing machines
  - nondeterministic Turing machines
  - enumerators
- all variants have same power as original
  - recognize the same class of languages
  - equivalent to original
  - robustness

Variants of Turing Machines

- to show robustness, vary transition function
  - instead of moving L or R, the head may stay put (S)
  - new transition function
    - $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R,S\}$
  - would this change allow additional languages to be recognized, thus increasing the power of Turing machines?
    - no, since we could convert S to two transitions
      - one that moves R
      - another that moves back L
- in general, to show variants are equivalent, we show how one can simulate another

Variants of Turing Machines

- multitape Turing machines
  - like regular Turing machine, but with several tapes
    - each tape has its own head for reading and writing
    - input appears on tape 1
    - other tapes start out blank
  - transition function changed to reading/writing/moving heads on multiple tapes simultaneously
    - $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R,S\}^k$
  - the expression

    $\delta(q_i,a_1,\ldots a_k) = (q_j,b_1,\ldots b_k,L,R,\ldots L)$ means
    - machine moves from q_i to q_j
    - heads 1 through k are, respectively,
      - reading symbols a_1 through a_k
      - writing symbols b_1 through b_k
      - moving L, R, or S

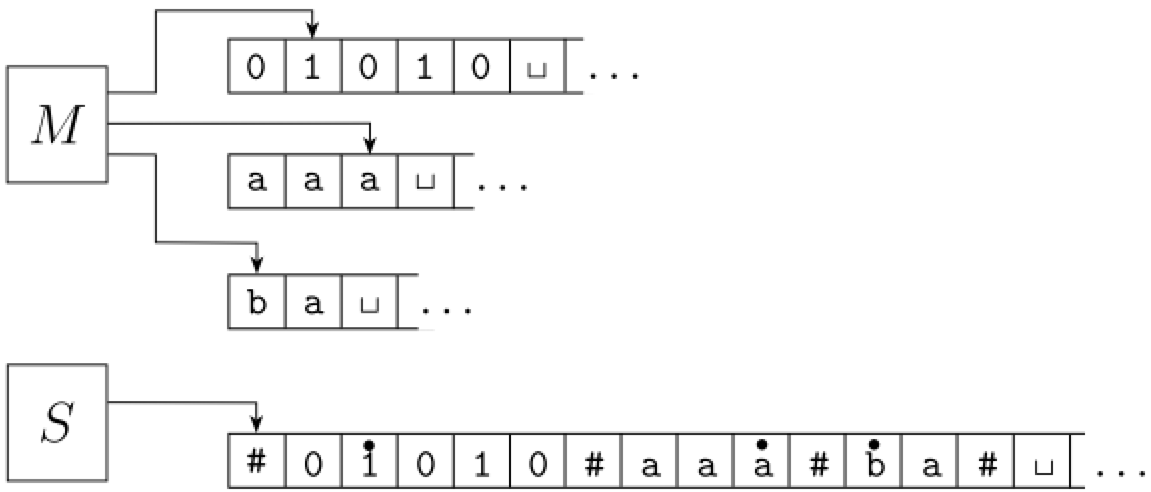
Variants of Turing Machines

- multitape Turing machines may seem more powerful, but we can prove they are equivalent to single-tape Turing machines by showing they recognize the same language

- proof: Show how to convert a multitape TM M to an equivalent single-tape TM S
  - simulate M with S
    - assume M has k tapes
    - S will simulate M by storing all information on one tape
      - uses # as a delimiter to separate contents of different tapes
      - keeps track of locations of different heads by placing a dot over the symbol where the head is currently located
      - # and dotted symbols are added to the tape alphabet Γ

Variants of Turing Machines

- proof: Show how to convert a multitape TM M to an equivalent single-tape TM S (cont.)
  - simulating M with S



Variants of Turing Machines

- proof: Show how to convert a multitape TM M to an equivalent single-tape TM S (cont.)

  S = "on input $w = w_1 \ldots w_n$

  1. S puts its tape into the format to represent k tapes

     $$\#\dot{w}_1 w_2 \;\cdots\; w_n \#\dot{\sqcup}\#\dot{\sqcup}\# \;\cdots\; \#.$$

  2. to simulate a single move, S scans from first # to last # to determine symbols under virtual heads
     - S makes a second pass to update tapes according to M's transition function
  3. if S moves a virtual head R and hits #
     - S writes a blank on this cell
       - shifts tape contents one unit right
     - continues with simulation."

Variants of Turing Machines

- a language is Turing-recognizable if and only if some multitape Turing machine recognizes it
  - proof:
    - a TRL must be recognized by an ordinary (single-tape) TM, which is a special case of a multitape TM
    - this proves one direction
    - other direction proven by previous proof

Variants of Turing Machines

- nondeterministic Turing machines
  - at any point in the computation, the machine may proceed
  - transition function has the form
    - $\delta: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L,R\})$
  - the computation of a nondeterministic Turing machine is a tree whose branches correspond to different possibilities for the machine
    - if some branch leads to the accept state → accept

Variants of Turing Machines

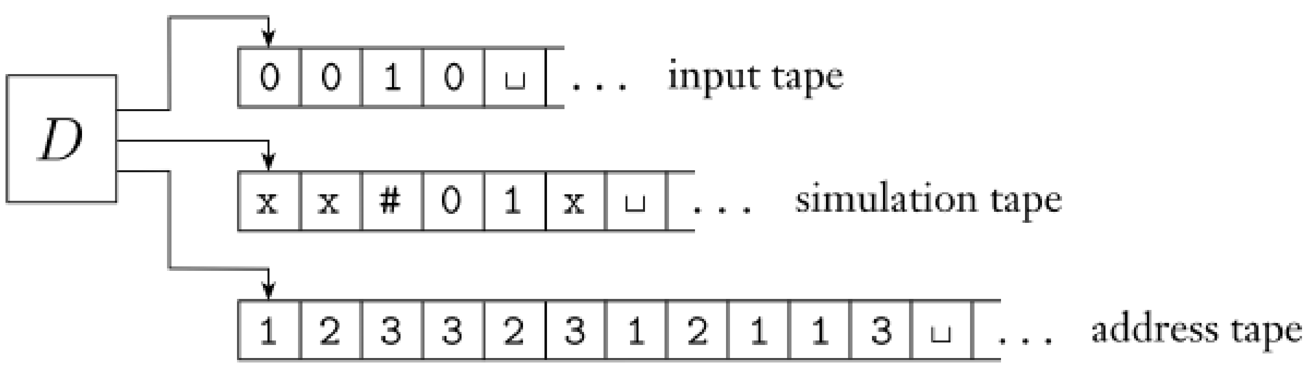
- every nondeterministic Turing machine has an equivalent deterministic Turing machine
  - proof idea:
    - simulate nondeterministic TM N with deterministic TM D
      - D will try all possible branches of N's computation
      - if D finds the accept state on a branch → accept
      - otherwise, D will not terminate

Variants of Turing Machines

- every nondeterministic Turing machine has an equivalent deterministic Turing machine (cont.)
  - proof idea:
    - view N's computation on w as a tree
      - each branch represents one of the branches of nondeterminism
      - each node is a configuration of N
      - root is the start configuration
      - D searches this tree for accepting configuration
        - depth-first search not good since a branch may be infinite (accept may be on another branch)
        - use breadth-first search instead

Variants of Turing Machines

- every nondeterministic Turing machine has an equivalent deterministic Turing machine (cont.)
  - proof:
    - D has three tapes (equivalent to having a single tape)
      - tape 1 has the input string and is never changed
      - tape 2 maintains a copy of N's tape on some branch of its nondeterministic computation
      - tape 3 keeps track of D's location in N



Variants of Turing Machines

- every nondeterministic Turing machine has an equivalent deterministic Turing machine (cont.)
  - proof:
    - more on tape 3
      - every node in the tree can have up to b children
      - b is largest set of choices in N's transition function
        - so $\Gamma_b = \{1, 2, \ldots, b\}$
        - e.g., address 231 means 2nd child of root, followed by 3rd child of next node, and 1st child of that node
      - empty string is address of the root

Variants of Turing Machines

- every nondeterministic Turing machine has an equivalent deterministic Turing machine (cont.)
  - proof:
    - D's computation
      1. initially, tape 1 contains input *w*; tapes 2 and 3 are empty
      2. copy tape 1 to tape 2 and initialize string on 3 to ε
      3. use tape 2 to simulate N
         - consult next symbol on tape 3 to determine choice from N's transition function
         - if no more symbols remain, or invalid choice, goto 4
         - also goto 4 if rejecting configuration encountered
         - if accepting configuration encountered → accept
      4. replace string on tape 3 with next string; goto 2

Variants of Turing Machines

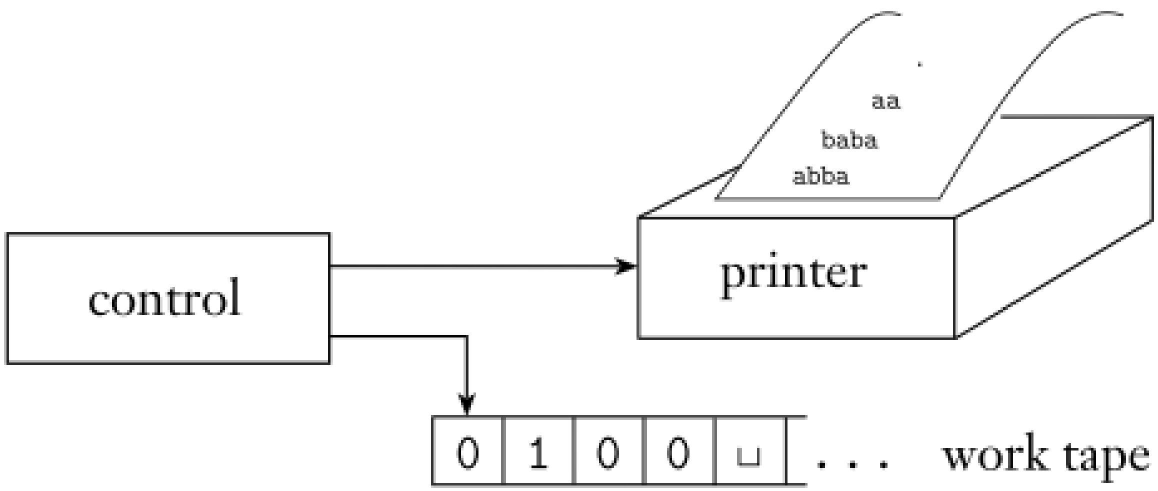
- a language is Turing-recognizable if and only if some nondeterministic Turing machine recognizes it
  - proof:
    - any deterministic TM is automatically a nondeterministic TM, which accounts for one direction of the proof
    - the other direction was proven previously

Variants of Turing Machines

- we can alter the previous proof so that if N always halts on all branches of computation, D will halt
- an NTM is called a decider if all branches halt on all inputs
- a language is decidable if and only if some NTM decides it

Variants of Turing Machines

- enumerators
  - Turing machine with attached printer
    - used as an output device to print strings



Variants of Turing Machines

- enumerators
  - enumerator E starts with blank input on work tape
    - if enumerator doesn't halt, list of strings may be infinite
    - language enumerated by E is all strings printed out
      - E may generate strings in any order, possibly with repetitions

Variants of Turing Machines

- a language is Turing-recognizable if and only if some enumerator enumerates it
  - proof
    - first show E enumerates language *A*, recognized by TM M
    - M = "on input w:
      1. run E. every time E outputs a string, compare it with w
      2. if w ever appears in the output of E → accept."
  - M accepts strings that appear on E's list

Variants of Turing Machines

- a language is Turing-recognizable if and only if some enumerator enumerates it (cont.)
  - proof
    - other direction: if TM M recognizes language A, we can construct an enumerator
    - say that $s_1, s_2, s_3, \ldots, s_i$ is a list of all possible strings in Σ^*
    - E = "ignore the input
      - repeat the following for i = 1, 2, 3, …
        - run M for i steps on each input $s_1, s_2, s_3, \ldots, s_i$
        - if any computations accept, print out corresponding s_j."

Variants of Turing Machines

- a language is Turing-recognizable if and only if some enumerator enumerates it (cont.)
  - if M accepts a particular s, it will eventually appear on the list generated by E
  - it will appear on the list infinitely many times because M runs from the beginning to the end for each first step
  - this procedure gives the effect of running M in parallel on all possible input strings

Variants of Turing Machines

- equivalence with other models
  - we have seen several TM variants equivalent to the original TM
    - many other models of general-purpose computation exist
    - some are quite different
    - all share the essential feature of TMs
      - unrestricted access to unlimited memory
        - unlike FAs and PDAs
      - all models with this feature are equivalent, as long as they satisfy reasonable requirements
        - e.g., perform only a finite amount of work in a single step

Variants of Turing Machines

- equivalence with other models (cont.)
  - consider analogous situation with programming languages
    - languages look different (e.g., LISP and Pascal)
    - same algorithms can be programmed in both
    - therefore, the two languages describe exactly the same class of of algorithms, as do all other reasonable programming languages
  - even though we can imagine many different computational models, the class of algorithms they describe is the same

The Definition of Algorithm

- algorithm: collection of simple instructions for carrying out some task
  - procedures or recipes
  - play important role in mathematics
    - ancient descriptions of algorithms for finding prime numbers, greatest common divisors, etc.
    - many algorithms today
  - algorithm not defined precisely until 20th century
    - previously, intuitive notion
    - needed for specific problems

The Definition of Algorithm

- Hilbert's Problems
  - Int'l Congress of Mathematics, Paris, 1900
  - 23 mathematical problems as challenges for the century
  - 10^{th} problem concerned algorithms

The Definition of Algorithm

- preliminary: review of polynomials
  - polynomial: sum of terms
    - each term is a product of variables and constant (or coefficient)
    - ex: $6 \cdot x \cdot x \cdot x \cdot y \cdot z \cdot z = 6x^3yz^2$
      - term with coefficient 6
    - ex: $6x^3yz^2 + 3xy^2 - x^3 - 10$
      - polynomial with 4 terms
      - we'll consider only coefficients that are integers
    - root of a polynomial: variable values when = 0
      - polynomial above has a root at $x = 5$, $y = 3$, $z = 0$
      - integral root because all integer values
        - some polynomials do not have an integral root

The Definition of Algorithm

- Hilbert's tenth problem
  - devise an algorithm that tests whether a polynomial has an integral root
  - in his words: a process in which it can be determined in a finite number of operations
    - assumed an algorithm existed
  - we now know no such algorithm exists
    - i.e., it is unsolvable
    - proving an algorithm does not exist required a clear definition of algorithm

The Definition of Algorithm

- Church-Turing thesis
  - 1936 paper by Church and Turing
  - Church: used notational system called λ-calculus to define algorithms
  - Turing: used Turing machines
  - two definitions were shown to be equivalent
  - connection between informal notation and precise definition is the Church-Turing thesis

Intuitive notion of algorithms	equals	*Turing machine algorithms*

The Definition of Algorithm

- Hilbert's tenth problem
  - in 1970, Matijasevic showed no algorithm exists for testing whether a polynomial has integral roots
  - rephrase the problem:
    - D = {p | p is a polynomial with an integral root}
    - in other words, is D decidable?

The Definition of Algorithm

- Hilbert's tenth problem
  - first, consider a simpler problem
    - D_1 = {p | p is a polynomial over x with an integral root}

  - TM M_1 that recognizes D_1:

    M_1 = "on input <p>: where p is a polynomial over x

    1. Evaluate p with x successively with values 0, 1, -1, 2, -2, 3, -3, … If at any point, the polynomial evaluates to 0, accept."

    - if p has an integral root, M_1 will find it and accept
    - if p does not, it will run forever

  - for D, M goes through all possible settings of variables to integral values

The Definition of Algorithm

- Hilbert's tenth problem
  - both M and M_1 are recognizers, but not deciders
    - we can convert M_1 to a decider for D_1 because we can restrict the roots to precalculated bounds
    - if a root is not found within these bounds, the machine rejects
    - Matijasevic's theorem shows that calculating such bounds is impossible

The Definition of Algorithm

- terminology for describing Turing machines
  - a Turing machine serves as a precise model for the definition of algorithms
    - Turing machines can describe any algorithm

The Definition of Algorithm

- standardize the way we describe Turing machine algorithms
  - what is the right level of detail? three possibilities
    - formal description
      - lists Turing machine's states, transition function, etc.
    - implementation description
      - use English prose to describe how the Turing machine moves the head and stores data on the tape
      - no details of states or transition functions
    - high-level description
      - use English prose to describe an algorithm
      - ignore implementation details
      - no mention of tape or head

The Definition of Algorithm

- so far, we have looked at formal and implementation-level descriptions
  - helps in understanding Turing machines

- high-level descriptions are sufficient

The Definition of Algorithm

- format and notation for Turing machines
  - input is always a string
    - can represent polynomials, graphs, grammars, automata, and combinations of these
    - Turing machine may decode these to be interpreted in any way desired
    - notation for encoding object O is <O>
    - for multiple objects $O_1, O_2, \ldots, O_k$, encoding is $\langle O_1, O_2, \ldots, O_k \rangle$

The Definition of Algorithm

- format and notation for Turing machines (cont.)
  - describe Turing machine algorithms with an indented segment of text within quotes
    - break the algorithm into stages
      - involving many individual steps
    - indicate block structure of the algorithm with further indentation
    - first line describes the input to the machine
      - if w, just a string
      - if <A>, machine must test whether the input properly encodes an object of the desired form
        - rejects if not

The Definition of Algorithm

- example: let *A* be the language of all strings representing undirected graphs that are connected
  - a graph is connected if every node can be reached from every other node by traveling along the edges of the graph
  - *A* = {<*G*> | *G* is a connected undirected graph}
  - high-level description of TM *M* that decides *A*

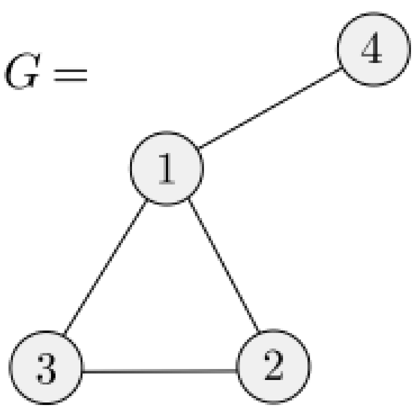
M = "On input <*G*>, the encoding of a graph *G*:

1. Select the first node of *G* and mark it.
2. Repeat the following stage until no new nodes are marked:
3. For each node in *G*, mark it if it is attached by an edge to a node that has already been marked.
4. Scan all the nodes of *G* to determine whether they all are marked. If they are, accept; otherwise, reject."

The Definition of Algorithm

- example: let *A* be the language of all strings representing undirected graphs that are connected (cont.)
  - implementation-level details
    - usually, we don't give this level of detail
    - first, how does <*G*> encode the graph as a string?
      - a list of nodes, followed by a list of edges
      - each node is a decimal number
      - each edge is a pair of decimal numbers representing the endpoints of an edge

$G =$



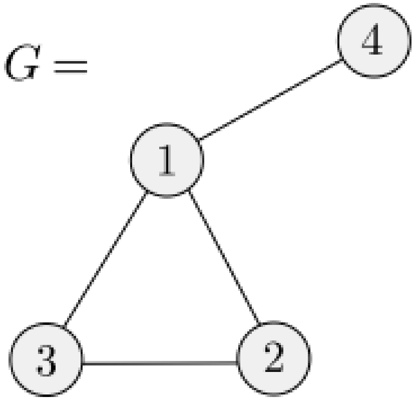
$\langle G \rangle =$

`(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`

The Definition of Algorithm

- example: let *A* be the language of all strings representing undirected graphs that are connected (cont.)
  - when M receives <G>, it checks for proper encoding
    - M scans tape to be sure there are two lists in the proper form
      - first list: distinct decimal numbers
        - contains no repetitions
        - use previous procedure for element distinctiveness
      - second list: pairs of decimal numbers
        - every node should also appear in the node list

$G =$



$\langle G \rangle =$

(1,2,3,4)((1,2),(2,3),(3,1),(1,4))

The Definition of Algorithm

- example: let A be the language of all strings representing undirected graphs that are connected (cont.)
  - after input check, M moves on to stage 1
  - stage 1: M marks first node with a dot on leftmost digit
  - stage 2: repeat the following until no new nodes are marked
    - stage 3: M scans the list of nodes to find an undotted node n_1 and flags it by marking it differently (underlining)
      - M then scans the list again to find a dotted node n_2 and underlines it, too
      - M scans the list of edges
      - for each edge, M tests whether the two underlined nodes n_1 and n_2 are the ones appearing on the edge
        - if so, M dots n_1, removes the underlines, and starts stage 2 again
        - if they aren't, M checks the next edge on the list
        - if there are no more edges, $\{n_1, n_2\}$ is not an edge of G
        - M moves the underline on n_2 to the next dotted node and calls it n_2
        - repeats check
        - if no more dotted nodes, n_1 is not attached to any dotted nodes
        - M sets the underline so that n_1 is the next undotted node and n_2 is the first dotted node and repeats
        - if there are no more dotted nodes, M has not been able to find any new nodes to dot, so it goes to stage 4
  - stage 4: scan the list of nodes to determine whether all are dotted
    - if so, enter accept state
    - otherwise, enter reject state