CS 341: Foundations of CS II

Marvin K. Nakayama
Computer Science Department
New Jersey Institute of Technology
Newark, NJ 07102

Chapter 3
Church-Turing Thesis

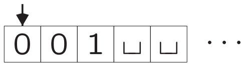
Contents

Previous Machines' Transitions Functions

- DFA, $\delta : Q \times \Sigma \rightarrow Q$
  - Reads input from left to right
  - Finite control (i.e., transition function) based on
    - current state,
    - current input symbol read.
- PDA, $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$
  - Has stack for extra memory
  - Reads input from left to right
  - Can read/write to memory (stack) by popping/pushing
  - Finite control based on
    - current state,
    - what's read from input,
    - what's popped from stack.

Turing machine (TM)

- Infinitely long **tape**, divided into cells, for memory
- Tape initially contains input string followed by all blanks ␣



- Tape **head** (↓) can move both **right** and **left**
- Can **read** from and **write** to tape
- Finite control based on
  - current state,
  - current symbol that head reads from tape.
- Machine has one **accept** state and one **reject** state.
- Machine can run forever: infinite **loop**.

Key Difference between TMs and Previous Machines

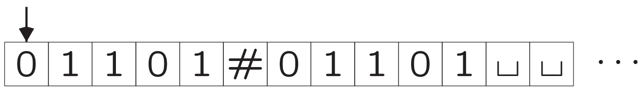
- Turing machine can both read from tape and **write** on it.
- Tape head can move both right and **left**.
- **Tape** is infinite and can be used for **storage**.
- Accept and reject states take **immediate effect**.

Example: Machine for recognizing language

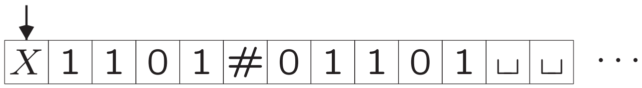
$$A = \{ s\#s \mid s \in \{0,1\}^* \}$$

Idea: Zig-zag across tape, crossing off matching symbols.

- Consider string 01101#01101 $\in A$.
- Tape head starts over leftmost symbol

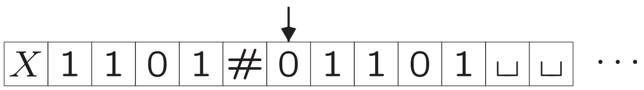


- Record symbol in control and overwrite it with X

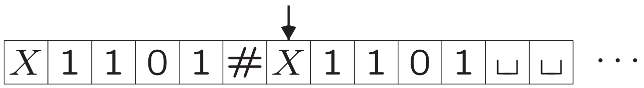


- Scan right: *reject* if blank “$\sqcup$” encountered before #

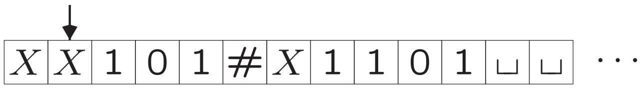
- When # encountered, move right one cell.



- If current symbol doesn’t match previously recorded symbol, *reject*.
- Overwrite current symbol with X

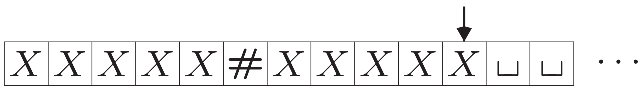


- Scan left, past # to X
- Move one cell right
- Record symbol and overwrite it with X

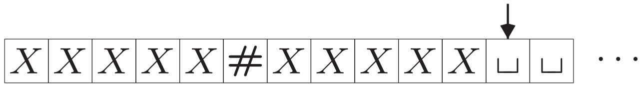


- Scan right past # to (last) X and move one cell to right ...

- After several more iterations of zigzagging, we have



- After all symbols left of # have been matched to symbols right of #, check for any remaining symbols to the right of #.
  - If blank $\sqcup$ encountered, *accept*.
  - If 0 or 1 encountered, *reject*.



- The string that is accepted or not by our machine is the original input string 01101#01101.

Description of TM M_1 for $\{ s\#s \mid s \in \{0,1\}^* \}$

$M_1 =$ "On input string $w \in \Sigma^*$, where $\Sigma = \{0, 1, \#\}$:

1. Scan input to be sure that it contains a single $\#$. If not, *reject*.
2. Zig-zag across tape to corresponding positions on either side of the $\#$ to check whether these positions contain the same symbol. If they do not, *reject*. Cross off symbols as they are checked off to keep track of which symbols correspond.
3. When all symbols to the left of $\#$ have been crossed off along with the corresponding symbols to the right of $\#$, check for any remaining symbols to the right of the $\#$. If any symbols remain, *reject*; otherwise, *accept*."

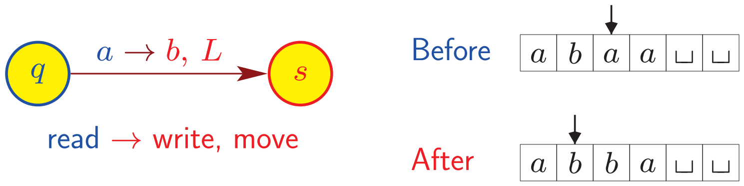
Formal Definition of Turing Machine

Definition: A **Turing machine** (TM) is a 7-tuple $M = (\, Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ q_{\text{accept}},\ q_{\text{reject}} \,)$, where

- Q is a finite set of **states**
- Σ is the **input alphabet** not containing blank symbol $\sqcup$
- Γ is **tape alphabet** with blank $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the **transition function**, where
  - L means move tape head one cell to left
  - R means move tape head one cell to right
- $q_0 \in Q$ is the **start state**
- $q_{\text{accept}} \in Q$ is the **accept state**
- $q_{\text{reject}} \in Q$ is the **reject state**, with $q_{\text{reject}} \neq q_{\text{accept}}$.

Transtion Function of TM

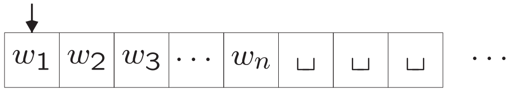
- Transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- $\delta(q, a) = (s, b, L)$ means
  - if TM
    - in state $q \in Q$, and
    - tape head reads tape symbol $a \in \Gamma$,
  - then TM
    - moves to state $s \in Q$
    - overwrites a with $b \in \Gamma$
    - moves head left (i.e., $L \in \{L, R\}$)



Start of TM Computation

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ begins computation as follows:

- Given input string $w = w_1 w_2 \cdots w_n \in \Sigma^*$ with each $w_i \in \Sigma$, i.e., w is a string of length n for some $n \geq 0$.
- TM begins in start state q_0
- Input string is on n leftmost tape cells



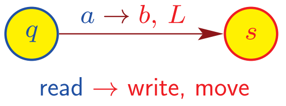
- Rest of tape contains blanks $\sqcup$
- Head starts on leftmost cell of tape
- Because $\sqcup \notin \Sigma$, first blank denotes end of input string.

TM Computation

When computation on TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ starts,

- TM M proceeds according to transition function

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$



- If M tries to move head off left end of tape,
  - then head remains on first cell.
- Computation continues until q_{accept} or q_{reject} is entered.
- Otherwise, M runs forever: infinite **loop**.
  - In this case, input string is neither accepted nor rejected.

Example: Turing machine M_2 recognizing language

$$A = \{\, 0^{2^n} \mid n \geq 0 \,\},$$

which consists of strings of 0s whose length is a power of 2.

Idea: The number k of zeros is a power of 2 iff successively halving k always results in a power of 2 (i.e., each result > 1 is never odd).

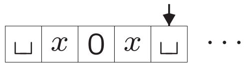
$M_2 =$ "On input string $w \in \Sigma^*$, where $\Sigma = \{0\}$:

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 end of the tape.
5. Go to stage 1."

Run TM M_2 with Input 0000

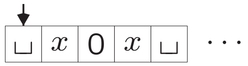
- Tape initially contains input 0000.

- Run stage 1: Sweep left to right across tape, crossing off every other 0.

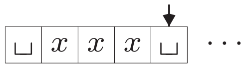


(Put $\sqcup$ in first cell to mark beginning of tape.)

- Run stage 4: Return head to left end of tape (marked by $\sqcup$).

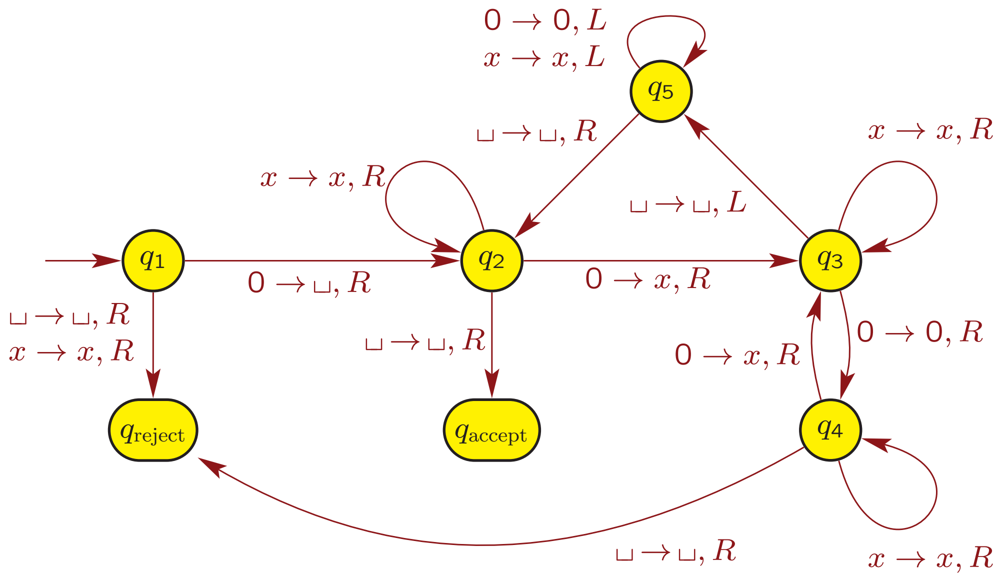


- Run stage 1: Sweep left to right across tape, crossing off every other 0.

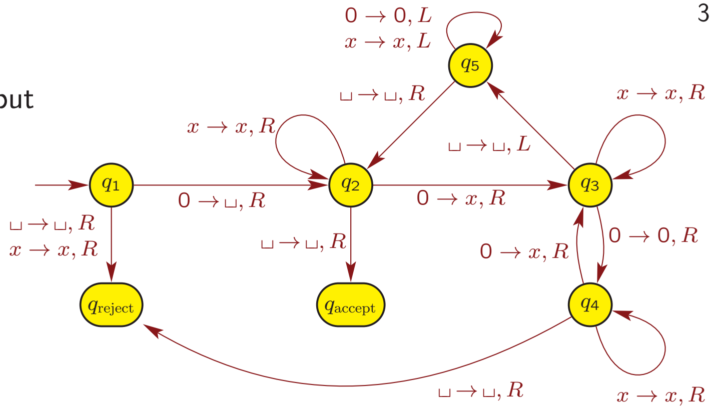


- Run stages 4 and 1: Return head to left end and scan tape.
- Run stage 2: If in stage 1 the tape contained a single 0, *accept*.

State Diagram of TM for $\{\, 0^{2^n} \mid n \geq 0 \,\}$



Run TM on input
$w = 0000$

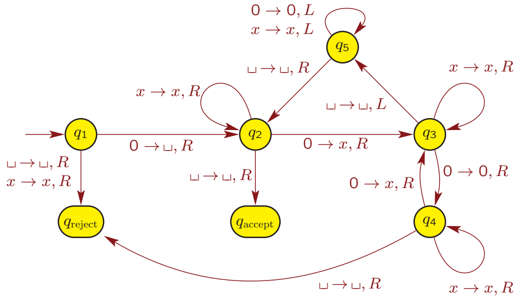


Step	State	Tape	Step	State	Tape
0	q_1	$\overset{\downarrow}{0}\ 0\ 0\ 0\ \sqcup\ \ldots$	4	q_3	$\sqcup\ x\ 0\ x\ \overset{\downarrow}{\sqcup}\ \ldots$
1	q_2	$\sqcup\ \overset{\downarrow}{0}\ 0\ 0\ \sqcup\ \ldots$	5	q_5	$\sqcup\ x\ 0\ \overset{\downarrow}{x}\ \sqcup\ \ldots$
2	q_3	$\sqcup\ x\ \overset{\downarrow}{0}\ 0\ \sqcup\ \ldots$	6	q_5	$\sqcup\ x\ \overset{\downarrow}{0}\ x\ \sqcup\ \ldots$
3	q_4	$\sqcup\ x\ 0\ \overset{\downarrow}{0}\ \sqcup\ \ldots$	⋮	⋮	⋮

TM for $\{ 0^{2^n} \mid n \geq 0 \}$

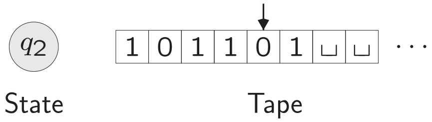
Turing machine $M_2 = (Q, \Sigma, \Gamma, \delta, q_1, q_{\text{accept}}, q_{\text{reject}})$, where

- $Q = \{q_1, q_2, q_3, q_4, q_5, q_{\text{accept}}, q_{\text{reject}}\}$
- $\Sigma = \{0\}$
- $\Gamma = \{0, x, \sqcup\}$
- Transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is specified in state diagram. For example,
  - $\delta(q_4, 0) = (q_3, x, R)$
  - $\delta(q_3, \sqcup) = (q_5, \sqcup, L)$
- q_1 is start state
- q_{accept} is accept state
- q_{reject} is reject state



TM Configurations

- Computation changes
  - current state
  - current head position
  - tape contents

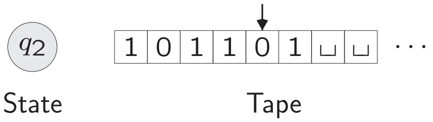


- **Configuration** provides "snapshot" of TM at any point during computation:
  - current state $q \in Q$
  - current tape contents $\in \Gamma^*$
  - current head location

TM Configurations

Configuration $1011q_201$ means

- current state is q_2
- LHS of tape is 1011
- RHS of tape is 01
- head is on RHS 0



Definition: a **configuration** of a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ is a string uqv with $u, v \in \Gamma^*$ and $q \in Q$, and specifies that currently

- M is in state q
- tape contains uv
- tape head is pointing to the cell containing the first symbol in v.

TM Transitions

Definition: Configuration C_1 **yields** configuration C_2 if the Turing machine can legally go from C_1 to C_2 in a single step.

- Specifically, for TM $M = (\Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, suppose
  - $u, v \in \Gamma^*$
  - $a, b, c \in \Gamma$
  - $q_i, q_j \in Q$
  - transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$.
- Then configuration uaq_ibv **yields** configuration $uacq_jv$ if

$$\delta(q_i, b) = (q_j, c, R).$$

TM Transitions

- Similarly, configuration uaq_ibv **yields** configuration uq_jacv if

$$\delta(q_i, b) = (q_j, c, L).$$

TM Transitions

- **Special case:** q_ibv yields q_jcv if

$$\delta(q_i, b) = (q_j, c, L)$$

  - If head is on leftmost cell of tape and tries to move left, then it stays in same place.

Remarks on TM Configurations

- Consider TM $M = (\,Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ q_{\text{accept}},\ q_{\text{reject}}\,)$.
- **Starting configuration** on input $w \in \Sigma^*$ is

$$q_0w$$

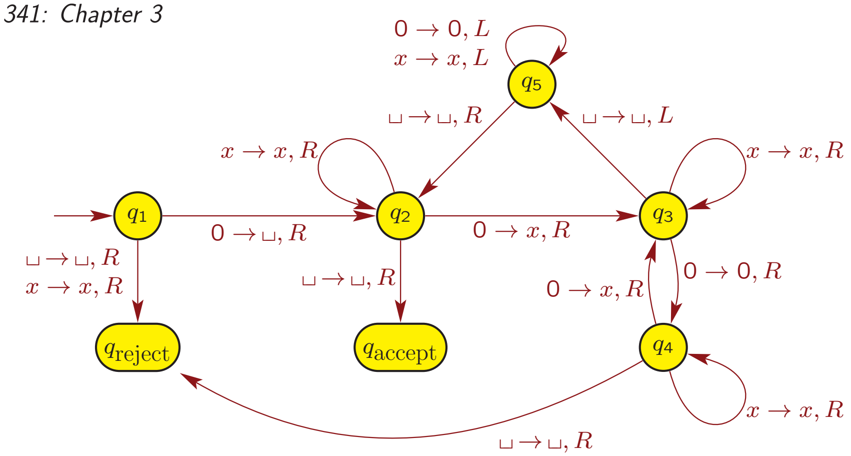
- An **accepting configuration** is

$$uq_{\text{accept}}v$$

  for some $u, v \in \Gamma^*$
- A **rejecting configuration** is

$$uq_{\text{reject}}v$$

  for some $u, v \in \Gamma^*$
- Accepting and rejecting configurations are **halting configurations**.
- Configuration uq_i is the same as $uq_i\sqcup$



On input 0000, get following sequence of configurations:

$$q_1 0000,\ \sqcup q_2 000,\ \sqcup x q_3 00,\ \sqcup x 0 q_4 0,\ \sqcup x 0 x q_3 \sqcup,\ \sqcup x 0 q_5 x,$$
$$\sqcup x q_5 0 x,\ \sqcup q_5 x 0 x,\ q_5 \sqcup x 0 x,\ \sqcup q_2 x 0 x,\ \sqcup x q_2 0 x,\ \sqcup x x q_3 x,$$
$$\sqcup x x x q_3 \sqcup,\ \sqcup x x q_5 x,\ \sqcup x q_5 x x,\ \sqcup q_5 x x x,\ q_5 \sqcup x x x,\ \sqcup q_2 x x x,$$
$$\sqcup x q_2 x x,\ \sqcup x x q_2 x,\ \sqcup x x x q_2 \sqcup,\ \sqcup x x x \sqcup q_{\text{accept}} \sqcup .$$

Formal Definition of TM Computation

- Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$.
- Input string $w \in \Sigma^*$.
- **Definition:** M **accepts** input w if there is a finite sequence of configurations $C_1, C_2, \ldots, C_k$ for some $k \geq 1$ with
  - C_1 is the starting configuration $q_0 w$
  - C_i yields C_{i+1} for all $i = 1, \ldots, k-1$
    - sequence of configurations obeys transition function δ
  - C_k is an accepting configuration $u q_{\text{accept}} v$ for some $u, v \in \Gamma^*$.
- **Definition:** The set of all input strings accepted by TM M is the language **recognized** by M and is denoted by $L(M)$.
  - Note that $L(M) \subseteq \Sigma^*$.

Turing-recognizable

Definition: Language A is **Turing-recognizable** if there is a TM M such that $A = L(M)$.

Remarks:

- Also called a **recursively enumerable** or **enumerable** language.
- On an input $w \notin L(M)$, the machine M can either
  - halt in a rejecting state, or
  - it can **loop indefinitely**
- How do you distinguish between
  - a very long computation and
  - one that will never halt?
- Turing-recognizable not practical because never know if TM will halt.

Turing-decidable

Definition: A **decider** is TM that halts on all inputs, i.e., never loops.

Definition: Language $A = L(M)$ is **decided** by TM M if on each possible input $w \in \Sigma^*$, the TM finishes in a halting configuration, i.e.,

- M ends in q_{accept} for each $w \in A$
- M ends in q_{reject} for each $w \notin A$.

Definition: Lang A is **Turing-decidable** if $\exists$ TM M that decides A.

Remarks:

- Also called a **recursive** or **decidable** language.
- Differences between *Turing-decidable* language A and *Turing-recognizable* language B
  - A has TM that halts on every string $w \in \Sigma^*$.
  - TM for B may loop on strings $w \notin B$.

Describing TMs

- It is assumed that you are familiar with TMs and with programming computers.
- Clarity above all:
  - high-level description of TMs is allowed; e.g.,

    $M =$ "On input string $w \in \Sigma^*$, where $\Sigma = \{0, 1\}$:
    1. Scan input . . . "
  - but it should not be used as a trick to hide the important details of the program.
- Standard tools: Expanding tape alphabet Γ with
  - separator "#"
  - dotted symbols $\overset{\bullet}{0}$, $\overset{\bullet}{a}$, to indicate "activity," as we'll see later.
  - Typical example: $\Gamma = \{0, 1, \#, \sqcup, \overset{\bullet}{0}, \overset{\bullet}{1}\}$

Example: Turing machine M_3 to decide language

$$C = \{\, a^i\, b^j\, c^k \mid i \times j = k \text{ and } i, j, k \geq 1 \,\}.$$

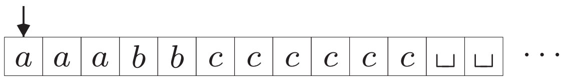
Idea: If i collections of j things each, then $i \times j$ things total.
TM: for each a, cross off j c's by matching each b with a c.

$M_3 =$ "On input string $w \in \Sigma^*$, where $\Sigma = \{a, b, c\}$:

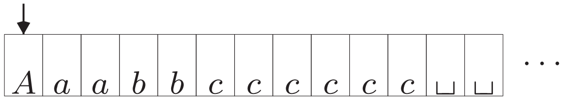
1. Scan the input from left to right to make sure that it is a member of $L(a^*b^*c^*)$, and *reject* if it isn't.
2. Return the head to the left-hand end of the tape
3. Cross off an a and scan to the right until a b occurs. Shuttle between the b's and the c's, crossing off each until all b's are gone. If all c's have been crossed off and some b's remain, *reject*.
4. Restore the crossed off b's and repeat stage 3 if there is another a to cross off. If all a's are crossed off, check whether all c's also are crossed off.
   If yes, *accept*; otherwise, *reject*."

Running TM M_3 on Input $a^3b^2c^6 \in C$

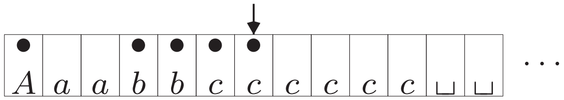
- Tape head starts over leftmost symbol



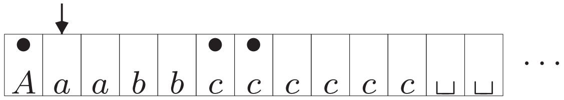
- Stage 1: Mark leftmost symbol and scan to see if input $\in L(a^*b^*c^*)$



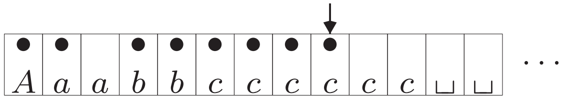
- Stage 3: Cross off one a and cross off matching b's and c's



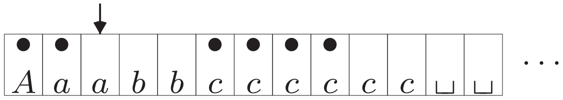
- Stage 4: Restore b's and return head to first a not crossed off



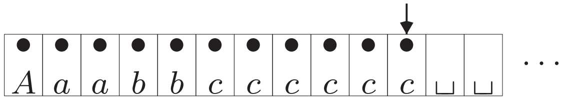
- Stage 3: Cross off one a and cross off matching b's and c's



- Stage 4: Restore b's and return head to first a not crossed off



- Stage 3: Cross off one a and cross off matching b's and c's

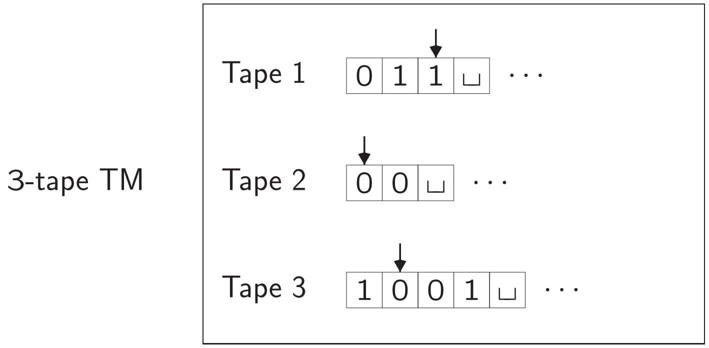


- Stage 4: If all a's crossed off, check if all c's crossed off.
- *accept*

TM Tricks

- **Question:** How to tell when a TM is at the left end of the tape?
- **One Approach:** Mark it with a special symbol.
- **Alternative method:**
    - remember current symbol
    - overwrite it with special symbol
    - move left
    - if special symbol still there, head is at start of tape
    - otherwise, restore previous symbol and move left.

Variant of TM: k-tape



- Each tape has its own head.
- Transitions determined by
    - current state, and
    - what all the heads read.
- Each head writes and moves independently of other heads.

---

k-tape Turing Machine

Definition: A **k-tape Turing machine**

$$M = (\, Q,\, \Sigma,\, \Gamma,\, \delta,\, q_0,\, q_{\text{accept}},\, q_{\text{reject}}\,)$$

has k different tapes and k different read/write heads:

- Q is finite *set of states*
- Σ is *input alphabet* (where $\sqcup \notin \Sigma$)
- Γ is *tape alphabet* with $(\{\sqcup\} \cup \Sigma) \subseteq \Gamma$
- q_0 is *start state* $\in Q$
- q_{accept} is *accept state* $\in Q$
- q_{reject} is *reject state* $\in Q$
- δ is *transition function*

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$$

where $\Gamma^k = \underbrace{\Gamma \times \Gamma \times \cdots \times \Gamma}_{k \text{ times}}$.

Multi-Tape TM

- Transition function

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$$

- Suppose

$$\delta(q_i, a_1, a_2, \ldots, a_k) = (q_j, b_1, b_2, \ldots, b_k, L, R, \ldots, L)$$

- Interpretation: If
    - machine is in state q_i, and
    - heads 1 through k read $a_1, \ldots a_k$,
- then
    - machine moves to state q_j
    - heads 1 through k write $b_1, \ldots, b_k$
    - each head moves left (L) or right (R) as specified.

Multi-Tape TM Equivalent to 1-Tape TM

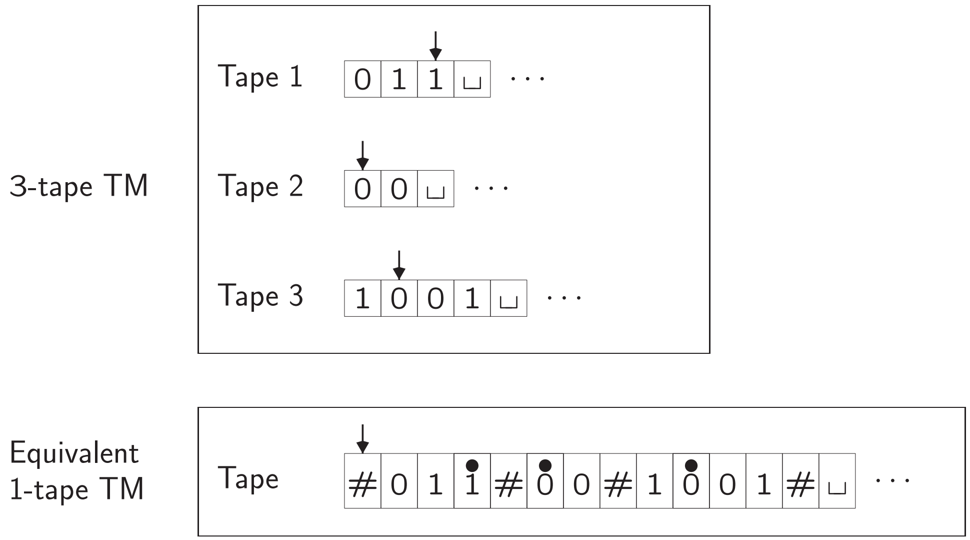
Theorem 3.13
For every multi-tape TM M, there is a single-tape TM M' such that $L(M) = L(M')$.

Remarks:

- In other words, for every multi-tape TM M, there is an **equivalent** single-tape TM M'.
- Proving and understanding this kind of robustness result is essential for appreciating the power of the TM model.
  - We will consider different variants of TMs, and show each has equivalent basic TM.

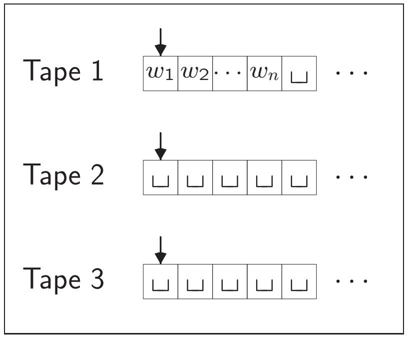
Basic Idea of Proof of Theorem 3.13

Simulate k-tape TM using 1-tape TM



Proof of Theorem 3.13

- Let $M_k = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ be a k-tape TM.

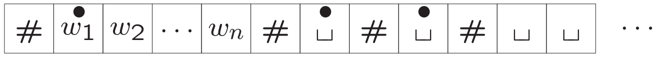


- Initially, M_k has
  - input $w = w_1 \cdots w_n$ on tape 1
  - other tapes contain only blanks ⊔
  - each head points to first cell.
- Construct 1-tape TM M_1 with expanded tape alphabet
  $\Gamma' = \Gamma \cup \dot{\Gamma} \cup \{\#\}$
  - Head positions are marked by dotted symbols in $\dot{\Gamma}$.

Proof of Theorem 3.13

On input $w = w_1 \cdots w_n$, the 1-tape TM M_1 does the following:

- First M_1 prepares initial string on single tape:



- For each step of M_k, TM M_1 scans tape **twice**
  1. Scans its tape from
     - first # (which marks left end of tape) to
     - $(k+1)$st # (which marks right end of used part of tape)

     to read symbols under “virtual” heads
  2. Rescans to write new symbols and move heads
     - If M_1 tries to move virtual head to the right onto #, then
       - ▲ M_k is trying to move head onto unused blank cell.
       - ▲ So M_1 has to write blank on tape and shift rest of tape right one cell.

Turing-recognizable $\Longleftrightarrow$ k-tape TM

From Theorem 3.13, we get the following:

Corollary 3.15
Language L is TM-recognizable if and only if some multi-tape TM recognizes L.

Nondeterministic TM

Definition: A **nondeterministic Turing machine** (NTM) M can have several options at every step. NTM is defined by a 7-tuple

$$M = (\, Q,\, \Sigma,\, \Gamma,\, \delta,\, q_0,\, q_{\text{accept}},\, q_{\text{reject}}\,),$$

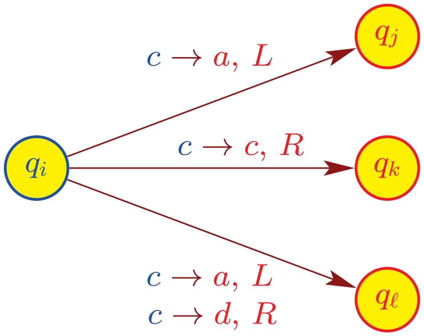
where

- Q is finite *set of states*
- Σ is *input alphabet* (without blank $\sqcup$)
- Γ is *tape alphabet* with $\{\sqcup\} \cup \Sigma \;\subseteq\; \Gamma$
- q_0 is *start state* $\in Q$
- q_{accept} is *accept state* $\in Q$
- q_{reject} is *reject state* $\in Q$
- δ is *transition function*

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

Transition Function δ of NTM

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

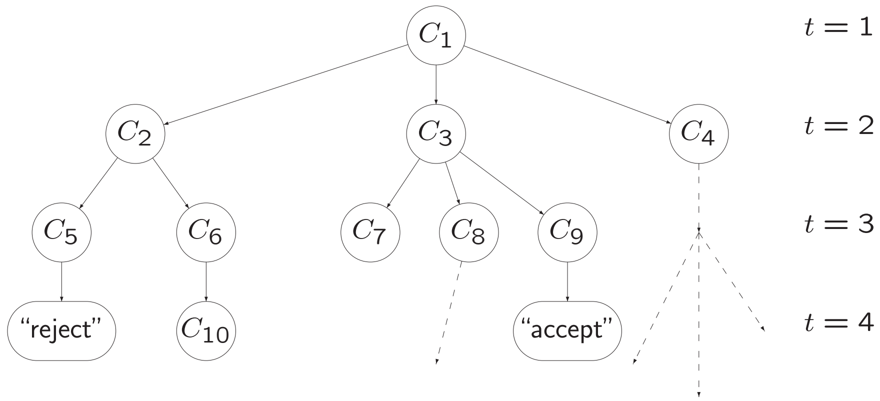


Multiple choices when in state q_i and reading c from tape:

$$\delta(q_i, c) = \{\, (q_j, a, L),\ (q_k, c, R),\ (q_\ell, a, L),\ (q_\ell, d, R)\, \}$$

Computing With NTMs

- On any input w, evolution of NTM represented by a **tree of configurations** (rather than a single chain).
- If $\exists$ (at least) one accepting leaf, then NTM **accepts**.



NTM Equivalent to TM

Theorem 3.16
Every nondeterministic TM N has an equivalent deterministic TM D.

Proof Idea:

- Build TM D to simulate NTM N on each input w.
- D tries all possible branches of N's tree of configurations.
- If D finds any accepting configuration, then it $accepts$ input w.
- If all branches reject, then D $rejects$ input w.
- If no branch accepts and at least one loops, then D $loops$ on w.

Proof of Equivalence of NTM and TM

On each input w, NTM N's computation is a **tree**

- Each branch is branch of nondeterminism.
- Each node is a **configuration** arising from running N on w.
- Root is starting configuration.
- TM D searches through tree to see if it has an accepting configuration.
  - Depth-first search (DFS) doesn't work. Why?
  - Breadth-first search (BFS) works.
- Tree doesn't actually exist.
  - So TM D needs to build tree while searching through it.

Proof of Equivalence of NTM and TM

Simulating TM D has 3 tapes

1. *Input tape*
   - contains input string w
   - never altered
2. *Simulation tape*
   - used as N's tape when simulating N's execution on some path in N's computation tree.
3. *Address tape*
   - keeps track of current location of BFS of N's computation tree.

Address Tape Works as Follows

- Every node in the tree has at most b children.
  - b is size of largest set of possible choices for N's transition fcn δ.
- Every node in tree has an **address** that is a string over the alphabet

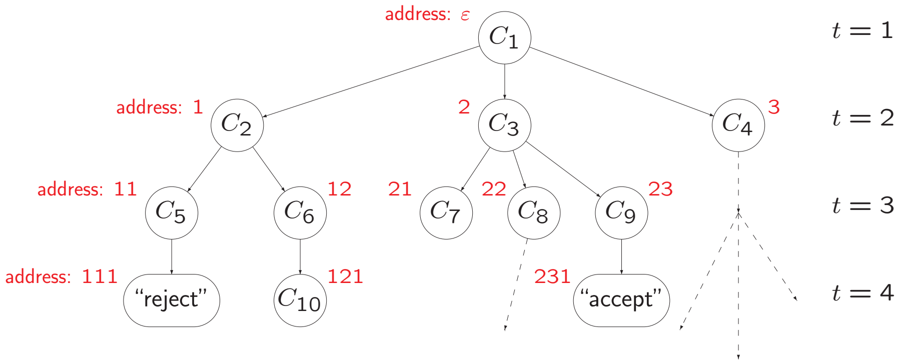
$$\Gamma_b = \{1, 2, \ldots, b\}$$

- To get to node with address **231**
  - start at root
  - take **second** branch
  - then take **third** branch
  - then take **first** branch
- Ignore meaningless addresses.
- Visit nodes in **BFS order** by listing addresses in Γ_b^* in **string order**:

$$\varepsilon,\ 1,\ 2,\ \ldots,\ b,\ 11,\ 12,\ \ldots,\ 1b,\ 21,\ 22,\ \ldots$$

Proof of Equivalence of NTM and TM

- "accept" configuration has address 231.
- Configuration C_6 has address 12.
- Configuration C_1 has address ε.
- Address 132 is meaningless.



TM D Simulating NTM N Works as Follows

1. Initially, input tape contains input string w.
   - Simulation and address tapes are initially empty.
2. Copy input tape to simulation tape.
3. Use simulation tape to simulate NTM N on input w on path in tree from root to the address on address tape.
   - At each node, consult next symbol on address tape to determine which branch to take.
   - *Accept* if accepting configuration reached.
   - Skip to next step if
     - symbols on address tape exhausted
     - nondeterministic choice invalid
     - rejecting configuration reached
4. Replace string on address tape with next string in Γ_b^* in string order, and go to Stage 2.

Remarks on TM Variants

Corollary 3.18
Language L is Turing-recognizable iff a nondeterministic TM recognizes it.

Proof.

- Every nondeterministic TM has an equivalent 3-tape TM
  1. input tape
  2. simulation tape
  3. address tape
- 3-tape TM, in turn, has an equivalent 1-tape TM by Theorem 3.13.

Remarks:

- k-tape TMs and NTMs are not more powerful than standard TMs:
- **The Turing machine model is extremely robust.**

TM Decidable $\Longleftrightarrow$ NTM Decidable

Definition: A nondeterministic TM is a **decider** if all branches halt on all inputs.

Remark: Can modify proof of previous theorem (3.16) so that if NTM N always halts on all branches, then TM D will always halt.

Corollary 3.19
A language is decidable iff some nondeterministic TM decides it.

Enumerators

Remarks:

- Recall: a language is **enumerable** if some TM recognizes it.
- But why *enumerable*?

Definition: An **enumerator** is a TM with a printer

- TM takes no input
- TM simply sends strings to printer
- may create infinite list of strings
- duplicates may appear in list
- *enumerates* a language

Enumerators

Theorem 3.21
Language A is Turing-recognizable iff some enumerator enumerates it.

Proof. Must show

1. $(\Leftarrow)$ If E enumerates language A, then some TM M recognizes A.
2. $(\Rightarrow)$ If TM M recognizes A, then some enumerator E enumerates A.

To show 1 $(\Leftarrow)$, given enumerator E,
build TM M for A using E as black box:

- $M =$ "On input string w,
  1. Run E.
  2. Every time E outputs a string, compare it to w.
  3. If w is output, $accept$."

Second Half of Proof of Theorem 3.21

We now show 2 $(\Rightarrow)$: If TM M recognizes A,
then some enumerator E enumerates A.

- Let $s_1, s_2, s_3, \ldots$ be an (infinite) list of all strings in Σ^*
- Given TM M, define E using M as black box as follows:
  - Repeat the following for $i = 1, 2, 3, \ldots$
    - Run M for i steps on each input $s_1, s_2, \ldots, s_i$.
    - If any computation accepts, print out corresponding string s
- Note that duplicates may appear.

"Algorithm" is Independent of Computation Model

- All reasonable variants of TM models are equivalent to TM:
  - k-tape TM
  - nondeterministic TM
  - enumerator
  - random-access TM: head can jump to any tape cell in one step.
- Similarly, all "reasonable" programming languages are equivalent.
  - Can take program in LISP and convert it into C, and vice versa.
- Notion of an **algorithm** is independent of computation model.

Algorithms

Muḥammad ibn Mūsā al-Khwārizmī
(c. 780 – c. 850)
source: wikipedia

What is an algorithm?

- Informally
  - a recipe
  - a procedure
  - a computer program
- Historically,
  - algorithms have long history in mathematics
  - but not precisely defined until 20th century
  - informal notions rarely questioned, but insufficient to show a problem has **no** algorithm.

Hilbert's 10th Problem

David Hilbert
(1862 – 1943)
source: wikipedia

In 1900, David Hilbert delivered a now-famous address

- Presented 23 open mathematical problems
- Challenge for the next century
- 10th problem concerned algorithms and polynomials

Polynomials

- A **term** is product of **variables** and constant integer **coefficient**:
$$6x^3yz^2$$
- A **polynomial** is a sum of terms:
$$6x^3yz^2 + 3xy^2 - x^3 - 10$$
- A **root** of a polynomial is an assignment of values to variables so that the value of the polynomial is zero.
- The above polynomial has a root at $(x, y, z) = (5, 3, 0)$.
- We are interested in **integral** roots.
- Some polynomials have integral roots; some don't.
  - Neither $21x^2 - 81xy + 1$ nor $x^2 - 2$ has an integral root.

Hilbert's 10th Problem

- **Problem:** Devise an algorithm that tests whether a polynomial has an integral root.
- In Hilbert's words:

> "to devise a process according to which it can be determined by a **finite** number of operations . . ."

- Hilbert seemed to assume that such an algorithm exists.
- However, Matijasevič proved in 1970 that no such algorithm exists.
- Mathematicians in 1900 couldn't have proved this.
  - No formal notion of an algorithm existed.
  - Informal notions work fine for constructing algorithms.
  - Formal notion needed to show no algorithm exists for a problem.

Church-Turing Thesis

Alonzo Church
(1903 – 1995)
source: wikipedia

Alan Turing
(1912 – 1954)
source: wikipedia

- Formal notion of algorithm developed in 1936
  - λ-calculus of Alonzo Church
  - Turing machines of Alan Turing
  - Definitions appear very different, but are equivalent.
- **Church-Turing Thesis**
  The informal notion of an **algorithm** corresponds exactly to a Turing machine that halts on all inputs.

Hilbert's 10th Problem

- For universe $\Omega = \{ p \mid p \text{ is a polynomial} \}$, consider language

$$D = \{ p \mid p \text{ is a polynomial with an integral root} \} \subseteq \Omega.$$

  - Since $6x^3yz^2 + 3xy^2 - x^3 - 10$ has an integral root at $(x, y, z) = (5, 3, 0)$,

$$6x^3yz^2 + 3xy^2 - x^3 - 10 \in D.$$

  - Since $21x^2 - 81xy + 1$ has no integral root,

$$21x^2 - 81xy + 1 \notin D.$$

- Hilbert's 10th problem asks whether this language is decidable.
  - i.e., Is there a TM that decides D?
- D is **not decidable**, but it is **Turing-recognizable**.

Hilbert's 10th Problem

- Consider simpler language of polynomials over single variable:

$$\begin{aligned} D_1 &= \{ p \mid p \text{ is a polynomial over } x \text{ with an integral root} \} \\ &\subseteq \{ p \mid p \text{ is a polynomial over } x \} \equiv \Omega_1 \end{aligned}$$

- D_1 is recognized by following TM M_1:
  - On input $p \in \Omega_1$, i.e., p is a polynomial over variable x
    1. Evaluate p with x set successively to values

$$0, 1, -1, 2, -2, 3, -3, \ldots.$$

    2. If at any point the polynomial evaluates to 0, *accept*.
- M_1 **recognizes** D_1, but **does not decide** D_1.
  - If p has an integral root, the machine eventually accepts.
  - If not, machine loops.

Hilbert's 10th Problem

- It turns out, though, that D_1 *is* decidable.
- Can show that the roots of p (over single variable x) lie between

$$\pm k \frac{c_{\text{max}}}{c_1}$$

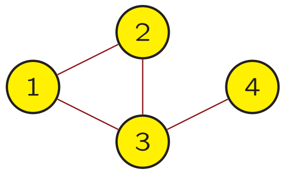
  where
  - k is number of terms in polynomial
  - c_{max} is maximum coefficient
  - c_1 is coefficient of highest-order term
- Thus, only have to check integers between $-k \frac{c_{\text{max}}}{c_1}$ and $k \frac{c_{\text{max}}}{c_1}$.
- Matijasevič proved such bounds don't exist for multivariate polynomials.

Encoding

- Input to a Turing machine is a string of symbols over an alphabet.
- But we want TMs (algorithms) that work on
  - polynomials
  - graphs
  - grammars
  - Turing machines
  - etc.
- Need to **encode** an *object* as a *string of symbols* over an alphabet.
- Can often do this in many reasonable ways.
- We sometimes distinguish between
  - an object X
  - its encoding $\langle X \rangle$.

Encoding an Undirected Graph

- Undirected graph G



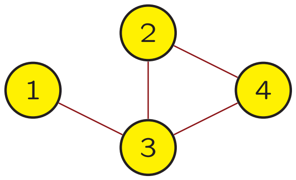
- One possible encoding

$$\langle G \rangle = \underbrace{(1,2,3,4)}_{\text{nodes}}\ \underbrace{(\,(1,2),\,(1,3),\,(2,3),\,(3,4)\,)}_{\text{edges}}$$

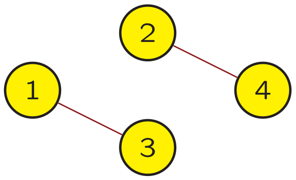
- In this encoding scheme, $\langle G \rangle$ of graph G is string of symbols over alphabet $\Sigma = \{0, 1, \ldots, 9, (,), ,\}$, where the string
  - starts with list of nodes
  - followed by list of edges

Connected Graphs

Definition: An undirected graph is **connected** if every node can be reached from any other node by travelling along edges.



Connected graph G_1



Unconnected graph G_2

Example: Let A be the language consisting of strings representing connected undirected graphs:

$$A = \{ \langle G \rangle \mid G \text{ is a connected undirected graph} \}.$$

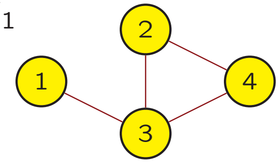
- $A \subseteq \Omega \equiv \{ \langle G \rangle \mid G \text{ is an undirected graph} \}$.
- $\langle G_1 \rangle \in A, \quad \langle G_2 \rangle \notin A$.

Decision Problems

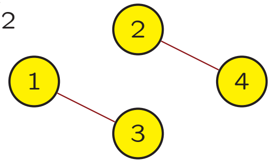
- **Decision problem:** (computational) question with YES/NO answer.
  - Answer depends on particular value of **input** to question.
- **Example:** Graph connectedness problem:

Is an undirected graph connected?

Graph G_1



Graph G_2



  - Input to question is from $\Omega \equiv \{ \langle G \rangle \mid G \text{ is an undirected graph} \}$.
  - For input $\langle G_1 \rangle$, answer is YES.
  - For input $\langle G_2 \rangle$, answer is NO.

Instance and Language of Decision Problem

- **Instance** of decision problem is **specific input value** to question.
  - Instance is encoded as string over some alphabet Σ.
  - **YES instance** has answer YES.
  - **NO instance** has answer NO.
- **Universe** Ω of a decision problem comprises **all instances**.
- **Language** of a decision problem comprises all its **YES instances**.
- **Example:** For graph connectedness problem,
  - Universe consists of (encodings of) **every** undirected graph G:

$$\Omega = \{ \langle G \rangle \mid G \text{ is an undirected graph} \}$$

  - Language A of decision problem

$$A = \{ \langle G \rangle \mid G \text{ is a } \textbf{connected} \text{ undirected graph} \}$$

is subset of universe; i.e., $A \subseteq \Omega$

Proving a Language is Decidable

- Recall for graph connectedness problem,

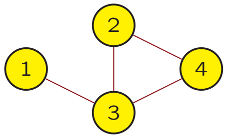
$$\begin{aligned} \Omega &= \{ \langle G \rangle \mid G \text{ is an undirected graph} \}, \\ A &= \{ \langle G \rangle \mid G \text{ is a } \textbf{connected} \text{ undirected graph} \} \subseteq \Omega. \end{aligned}$$

- To prove A is decidable language, need to show $\exists$ TM that **decides** A.
- For a TM M to **decide** A, the TM must
  - take any instance $\langle G \rangle \in \Omega$ as input
  - halt and **accept** if $\langle G \rangle \in A$
  - halt and **reject** if $\langle G \rangle \notin A$ (i.e., never loops indefinitely)

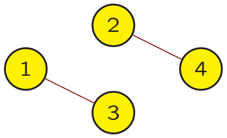
TM to Decide if Graph is Connected

$$\begin{aligned} A &= \{ \langle G \rangle \mid G \text{ is a } \textbf{connected} \text{ undirected graph} \} \\ &\subseteq \{ \langle G \rangle \mid G \text{ is an undirected graph} \} \equiv \Omega \end{aligned}$$

Graph G_1



Graph G_2



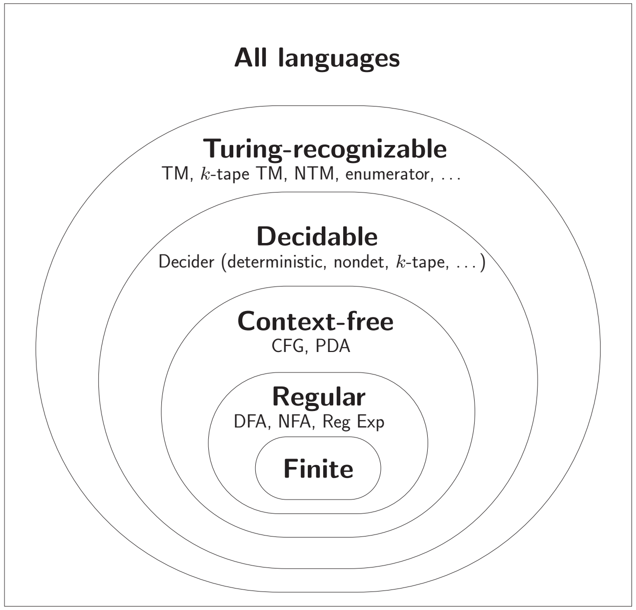
```
M = "On input ⟨G⟩ ∈ Ω, where G is an undirected graph:
     0. Check if ⟨G⟩ is a valid graph encoding. If not, reject.
     1. Select first node of G and mark it.
     2. Repeat until no new nodes marked:
     3.     For each node in G, mark it if it's attached by an
            edge to a node already marked.
     4. Scan all nodes of G to see whether they all are marked.
        If they are, accept; otherwise, reject."
```

TM M for Deciding Language A

For TM M that decides $A = \{ \langle G \rangle \mid G \text{ is a connected undirected graph} \}$

- Stage 0 checks that input $\langle G \rangle \in \Omega$ is valid graph encoding, e.g.,
  - two lists
    - first is a list of numbers
    - second is a list of pairs of numbers
  - first list contains no duplicates
  - every node in second list appears in first list
- Stages 1–4 then check if G is connected.
- When defining a TM, we often do not explicitly include stage 0 to check if the input is a valid encoding.
  - Instead, the check is often only implicitly included.

Hierarchy of Languages (so far)



All languages

Turing-recognizable
TM, k-tape TM, NTM, enumerator, ...

Decidable
Decider (deterministic, nondet, k-tape, ...)

Context-free
CFG, PDA

Regular
DFA, NFA, Reg Exp

Finite

Examples

???

???

$\{ 0^n 1^n 2^n \mid n \geq 0 \}$

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

$(0 \cup 1)^*$

$\{ 110, 01 \}$

Summary of Chapter 3

- Turing machines
  - tape head can move right and **left**
  - tape head can read and **write**
- TM computation can be expressed as sequence of configurations
- Language is **Turing-recognizable** if some TM recognizes it
  - But TM **may loop forever** on input string not in language
- Language is **Turing-decidable** if a TM decides it (must always halt)
- Variants of TM (k-tape, nondeterministic, etc.) have equivalent TM
- Church-Turing Thesis
  - Informal notion of algorithm is same as deciding by TM.
- Hilbert's 10th problem undecidable.
- Encoding TM input and decision problems.