Midterm Announcements

- Topics on midterm:

  Beginning of course through Monday 2/19 lecture

  ⇒ Includes Turing reductions, but not the type where you need to construct another machine
- Midterms from previous terms have been released. Format will be similar, but see HW and discussion worksheets to review all topics
- You may bring one double-sided 8.5 x 11 study sheet, that you prepare
- Review of past exams today 6-8pm BBB 1670 with Eric K.
- Wednesday 3/6:
  - No lecture
  - Office hours end at 5pm
  - Midterm 7-9pm

Techniques/concepts

Algorithmic techniques

- Potential method
- Divide-and-Conquer + Master Theorem
- Dynamic Programming
- Greed + Induction/Exchange

Models of Computation:

- DFAs
- Turing machines + Church-Turing thesis
- Terminology: countable vs uncountable, language, (un)decidable

Techniques for proving undecidability

- Diagonalization/paradox
- Reduction

Reminder of problems + algorithms from class

- **Potential method:** GCD (Euclid)
- **Divide-and-conquer:** sorting (mergesort), closest pair, integer multiplication (Karatsuba)
- **Dynamic programming:** weighted task selection, LIS, LCS, knapsack, SSSP (Bellman-Ford), APSP (Floyd-Warshall)
- **Greedy:** unweighted task selection, MST (Kruskal)
- **Countable vs uncountable sets:** integers, rationals, reals, TMs, TM inputs, languages
- **Undecidable languages:** L_{BARBER}, L_{ACC}, L_{HALT}

Some reference slides copied from past lectures

Potential Method

Intuitively, a **potential function argument** says:
If I start with a finite amount of water in a leaky bucket, then eventually water must stop leaking out.

Ingredients of the argument:

1. Define the “unit of time” e.g. one iteration of an algorithm
2. Define how we measure the amount of water in the bucket. This is the **potential function S_i** $\leftarrow$ amount of water in bucket at timestep i
3. Prove that the S_0 is finite and S_i can never be negative
4. Prove that the bucket “leaks quickly”. I.e. that S_i decreases by at least some fixed amount per unit time.
5. Use this to upper bound the total number of units of time.

Overview: Divide-and-Conquer Algorithms

Main Idea:

1. **Divide** the input into smaller sub-problems
2. **Conquer:** solve each sub-problem recursively and combine their solutions

Designing the Algorithm + Proving Correctness: an "art"

- Depends on problem structure, ad-hoc, creative

Running time Analysis: "mechanical"

- Express runtime using a recurrence
- Can often solve using the "Master Theorem"

Solving Recurrences

The Master Theorem

Consider the following recurrence relation where $k \geq 1$, $b > 1$, $d \geq 0$, $w \geq 0$ are constants (they do not depend on n):

$$T(n) = kT(n/b) + O(n^d \log^w n)$$
$$T(1) = O(1)$$

Then:

$$T(n) = \begin{cases} O(n^d \log^w n) & \text{if } k < b^d \\ O(n^d \log^{w+1} n) & \text{if } k = b^d \\ O(n^{\log_b k}) & \text{if } k > b^d \end{cases}$$

The DP Recipe

1. Write recurrence ← usually the trickiest part
2. Size of table: How many dimensions? Range of each dimension?
3. What are the base cases?
4. To fill in a cell, which other cells do I look at? In which order do I fill the table?
5. Which cell(s) contain the final answer?
6. Running time = (size of table) · (time to fill each entry)
7. To reconstruct the solution (instead of just its size) follow arrows from final answer to base case

General strategy commonly used for analyzing greedy algorithms:

Proof by induction using an "**exchange**" argument

The idea: Show that we can transform any **optimal solution** into the **solution given by our algorithm** by **exchanging** each piece of it out one-by-one without increasing the cost.

Key part of proof: **Exchange** shows that my greedy choice is **safe** i.e. it is in some optimal solution.

Induction formalizes the idea that *each successive choice* is **safe**.

String notation

Alphabet: A nonempty finite set Σ of symbols.

$\Sigma = \{0,1\}$ is a popular choice.

String: A finite sequence of 0 or more symbols.
(or "word")

The empty string is denoted ε.

For any $a \in \Sigma$:

a^k means k a's

a^* means ≥0 a's

a^+ means ≥1 a's

Σ^k means all strings over Σ of length k.

Σ^* means **all** (finite) strings over Σ.

Σ^+ means all nonempty (finite) strings over Σ

For any $a,b \in \Sigma$: $a|b$ means a OR b

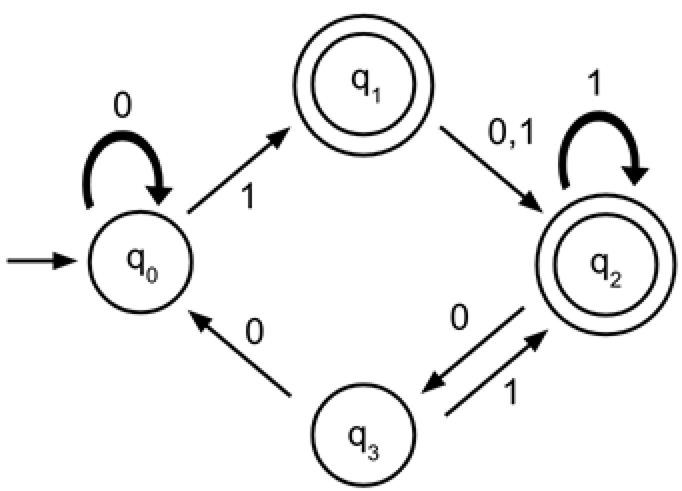
Language: A collection of strings.

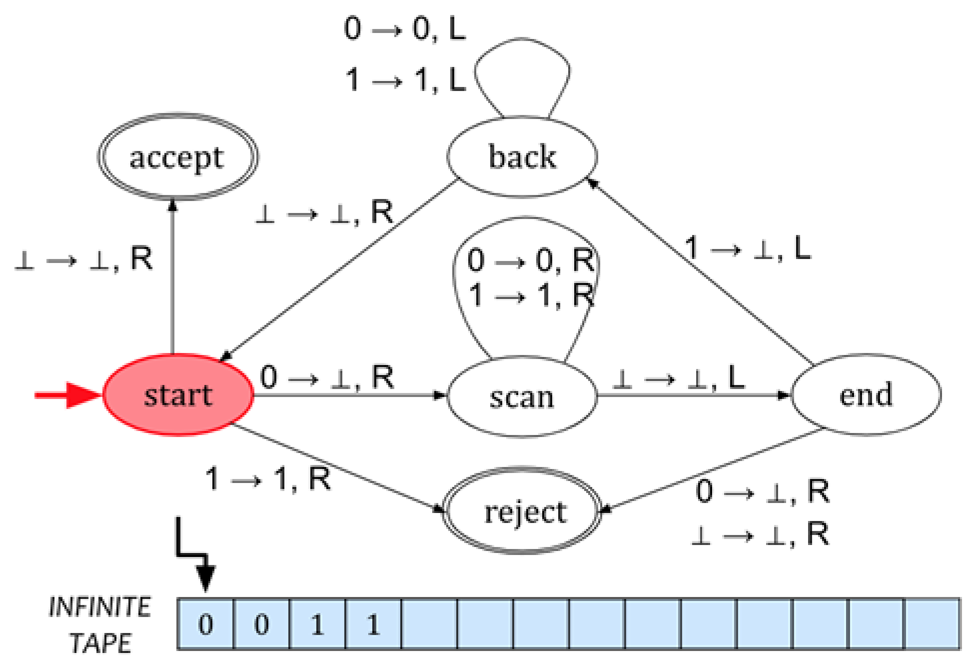
I.e. any subset $L \subseteq \Sigma^*$.

The empty language is denoted Ø.

DFA

Turing Machine

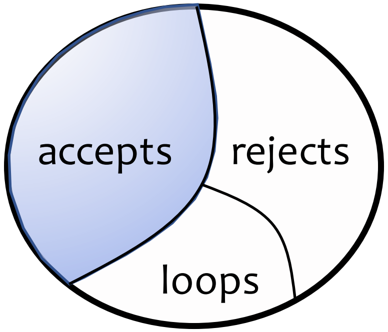




Undecidability

Question: What are the possible outcomes of a TM **M**?

Answer: **M** either (i) accepts, (ii) rejects, *or* (iii) it *"loops" (forever)*



The language of a TM is the set of strings it accepts:
L(M) = {x : M accepts x}

Definition: A Turing Machine **M** ***decides*** a language L if it:

1. *accepts* every string in L, and
2. *rejects* every string not in L

(and never loops forever)

A *language* L is ***decidable*** if there is a TM that decides L.
Otherwise L is ***undecidable***.

Turing Reduction from A to B (denoted $A \leq_T B$):

"We can use a black-box decider for B
as a subroutine to decide A."

M_A:

$M_B(x)$

What it implies:

1. If B is decidable then A is decidable.
2. Contrapositive: If A is undecidable then B is undecidable.

You can even invoke M_B multiple times inside of M_A if you want

input to M_A

M_A: Decider for A

1. do some stuff
2. give input to M_B

M_B: Decider for B

3. get output from M_B
4. do some more stuff

output of M_A

"Problem B is at least as hard as Problem A"

$A[1 \ldots n]$ 4 3 7 2 9 3 1 → 10, 6

H(3)

$H(i)$: soln ending exactly at index i

$H(1)$	= 3	3
$H(2)$	= 7	7
$H(3)$	= 9	9
$H(4)$	1	9
$H(5)$	9	12
$H(6)$	~~12~~	13

$H(1) = A[1]$

$$H(i) = A[i] + \max_{\substack{A[j] > A[i] \\ 1 \le j < i}} \{H(j)\} \cup \{0\}$$

where we define 0 if doesn't exist.

Final answer: Max

$$LCS(S1[1..\underset{i}{N}],\ S2[1..\underset{j}{M}]) =$$