CS420, Spring 2021

Last Lecture

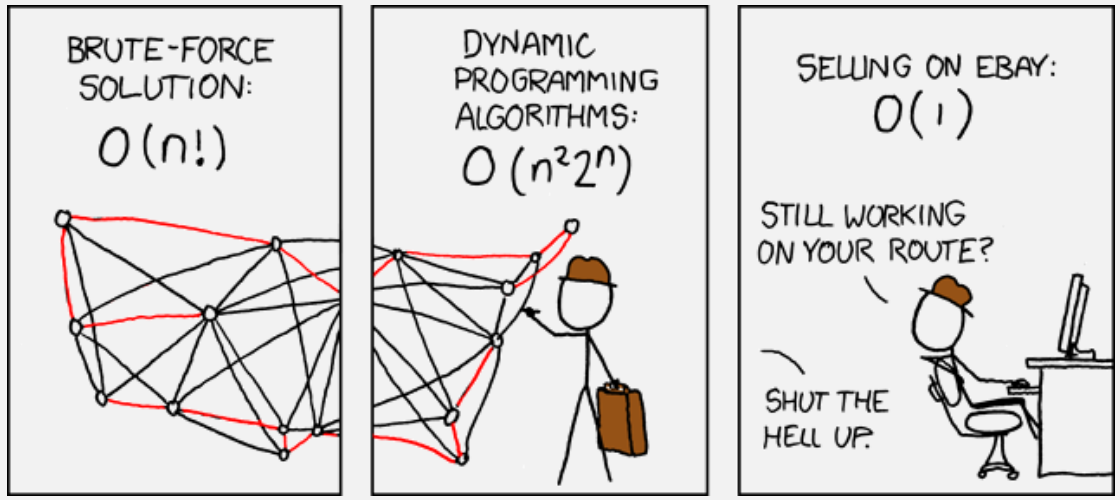
Wednesday, May 12, 2021

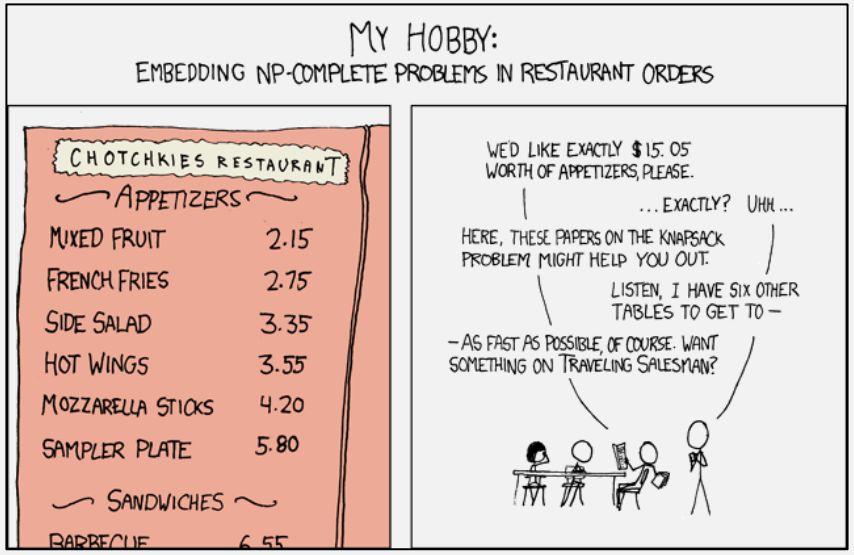
Announcements

- HW12 extended
  - due Friday 5/14 11:59pm EST
- This week is the last week of Office Hours
- Fill out course evaluation
  - Will have time at end of this lecture

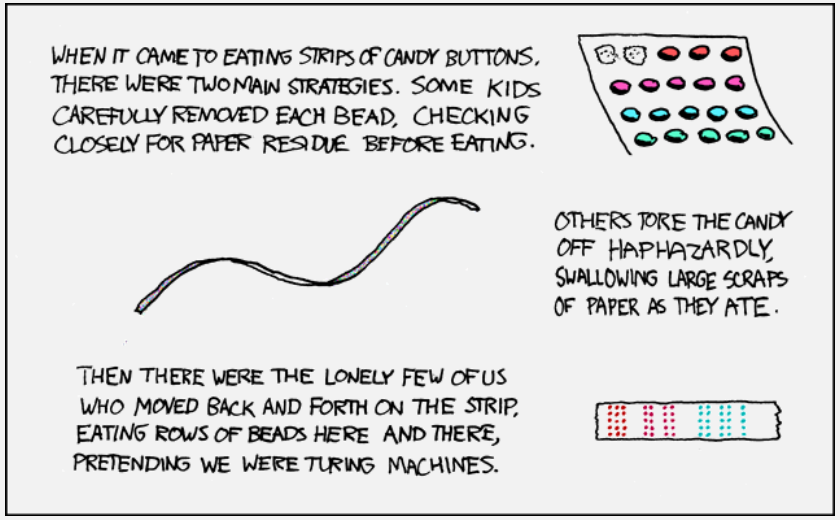
Flashback (Lecture 1): Why Take CS420?

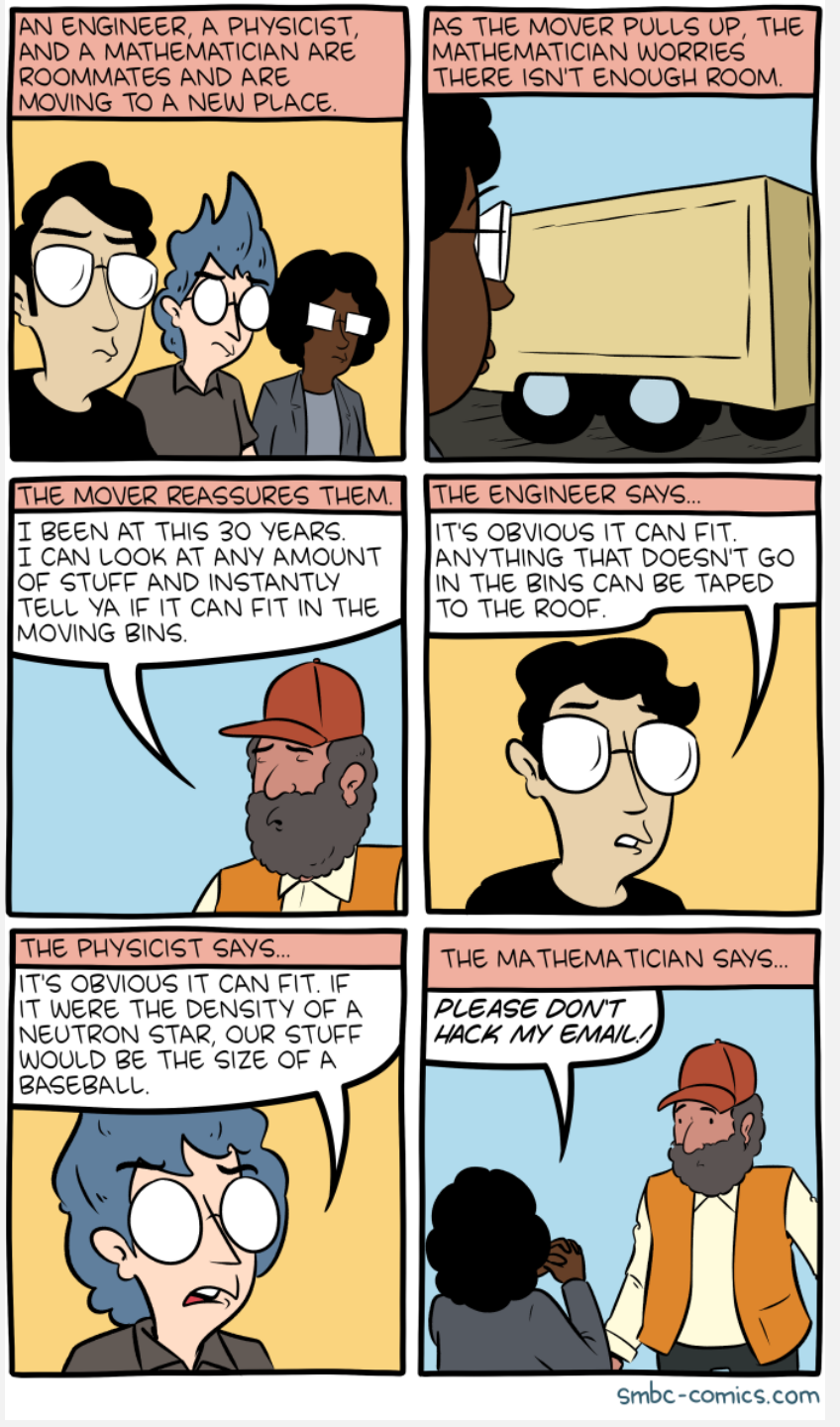
To be able to understand CS jokes!


BRUTE-FORCE SOLUTION:
O(n!)
DYNAMIC PROGRAMMING ALGORITHMS:
O(n²2ⁿ)
SELLING ON EBAY:
O(1)
STILL WORKING ON YOUR ROUTE?
SHUT THE HELL UP.

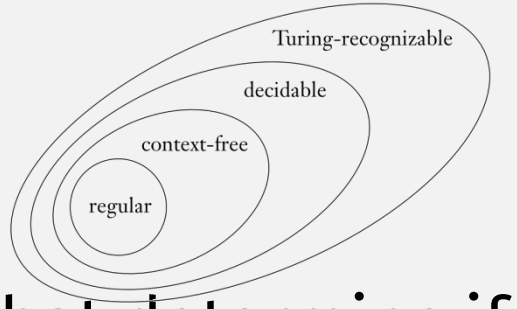

MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS
CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
BARBECUE 6.55
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO—
—AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

POLL


WHEN IT CAME TO EATING STRIPS OF CANDY BUTTONS, THERE WERE TWO MAIN STRATEGIES. SOME KIDS CAREFULLY REMOVED EACH BEAD, CHECKING CLOSELY FOR PAPER RESIDUE BEFORE EATING.
OTHERS TORE THE CANDY OFF HAPHAZARDLY, SWALLOWING LARGE SCRAPS OF PAPER AS THEY ATE.
THEN THERE WERE THE LONELY FEW OF US WHO MOVED BACK AND FORTH ON THE STRIP, EATING ROWS OF BEADS HERE AND THERE, PRETENDING WE WERE TURING MACHINES.


AN ENGINEER, A PHYSICIST, AND A MATHEMATICIAN ARE ROOMMATES AND ARE MOVING TO A NEW PLACE.
AS THE MOVER PULLS UP, THE MATHEMATICIAN WORRIES THERE ISN'T ENOUGH ROOM.
THE MOVER REASSURES THEM.
I BEEN AT THIS 30 YEARS. I CAN LOOK AT ANY AMOUNT OF STUFF AND INSTANTLY TELL YA IF IT CAN FIT IN THE MOVING BINS.
THE ENGINEER SAYS...
IT'S OBVIOUS IT CAN FIT. ANYTHING THAT DOESN'T GO IN THE BINS CAN BE TAPED TO THE ROOF.
THE PHYSICIST SAYS...
IT'S OBVIOUS IT CAN FIT. IF IT WERE THE DENSITY OF A NEUTRON STAR, OUR STUFF WOULD BE THE SIZE OF A BASEBALL.
THE MATHEMATICIAN SAYS...
PLEASE DON'T HACK MY EMAIL!
smbc-comics.com

CS420: A Course About Classifying Languages

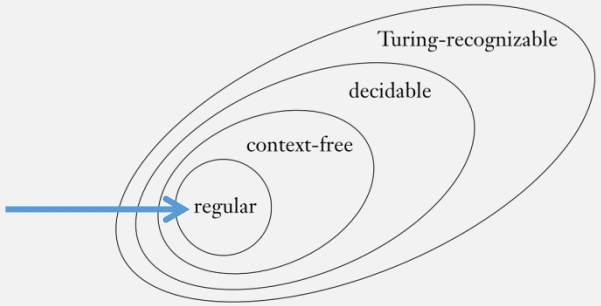


- A language = a set of strings

- We classify languages according to the machines that determine if some string is a member of the language
  - i.e., the computational "power" needed to recognize the language

- Practical applications:
  - Knowing if a "machine" is sufficient for a task
    - Don't try to parse HTML/XML with regular expressions
    - Don't try to parse a non-CFL with a PDA
  - Knowing what is "solvable" in practice
    - Don't try to brute force compute HAMPATH for a million node graph
  - Exploring the limits of computation
    - Don't try to compare if two CFGs are equivalent

Chapter 1: Regular Languages



- A regular language is one that is recognized by a DFA:

DEFINITION 1.5

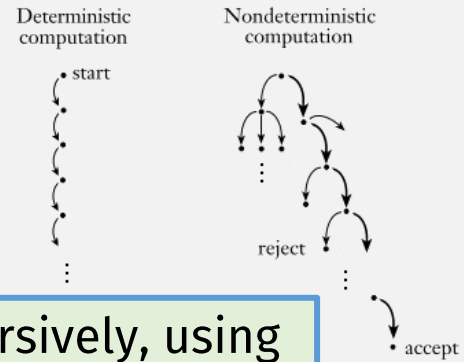
A *finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the *states*,
2. Σ is a finite set called the *alphabet*,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the *transition function*,[1]
4. $q_0 \in Q$ is the *start state*, and
5. $F \subseteq Q$ is the *set of accept states*.

- Or recognized by an NFA:

COROLLARY 1.40

A language is regular if and only if some nondeterministic finite automaton recognizes it.



- Or described by a regular expression

THEOREM 1.54

A language is regular if and only if some regular expression describes it.

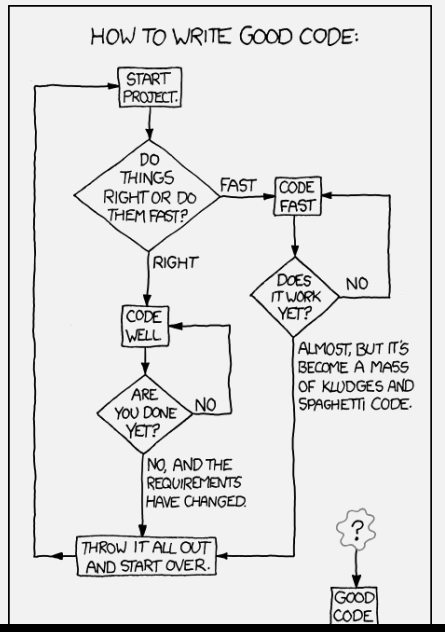
Defined recursively, using three closed operations:
- **Union**
- **Concatentation**
- **Kleene star**

Creating Machines = Programming

HW 1, Problem 1

Our first task is to design a data representation for this mathematical DFA definition.

You may use objects, structs, or anything else available in your language.



DEFINITION 1.5

A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ is the ***transition function***,[1]
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subseteq Q$ is the ***set of accept states***.

```
;; A State is a Symbol

;; A DFA is a dfa struct instance consisting of:
;; - Q : [Listof State] - all the machine states
;; - Σ : [Listof Char] - all possible input chars
;; - δ : fn mapping State Char → State - valid state transitions
;; - q0 : State - the start state
;; - F : [Listof State] - the accept states
(struct dfa (states Σ δ q0 F))
```

- All programs have a “math” definition
- In industry, this “math” is called a “spec”, or “requirement”
- It specifies what the program is supposed to do
  - I.e., it establishes criteria for the program to be correct

Mathematical "Computation" → "Run" Fn

HW 2, Problem 1

Write a "run" predicate (a function or method that returns true or false) that takes two arguments, an instance of your DFA representation (as defined in A Data Representation for DFAs) and a string, and "runs" the string on the DFA.

The function should return true if the DFA accepts the string, and false otherwise.

FORMAL DEFINITION OF COMPUTATION

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M ***accepts*** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

1. $r_0 = q_0$,
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots, n-1$, and
3. $r_n \in F$.

Math / spec

A program for the spec

```
;; accept? : DFA String → Bool
(define (accept? M w)
  (match (dfa _ _ δ q0 F) M)
  (let LOOP ([current-state q0]
             [input w])
    (if (empty? input)
        (member current-state F)
        (LOOP (δ current-state (first input))
              (rest input)))))
```

Mathematical "Language" → "Lang" Fn

HW 2, Problem 2

- Using your predicate from Simulating Computation for DFAs, write a function or method that, given a DFA, prints all strings in the language that it recognizes, up to strings of length 5, one per line.

We say that M ***recognizes language*** A if $A = \{w | \ M \text{ accepts } w\}$.

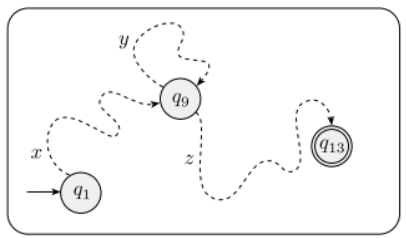
```
;; lang : DFA Int → [Listof String]
;; returns all strings accepted by dfa, up to length len
(define (lang M len)
  (filter (accept? M) (all-possible (dfa-Σ M) len)))
```

The Limits of Regular Languages

- Regular languages can't have dependencies between parts
  - Because DFAs/NFAs have only finite memory (their states)

- E.g. of a nonregular language, $\{0^n 1^n \mid n \geq 0\}$
  - This lang in essence is "paren matching"
  - Thus, the syntax of nearly every programming lang is also nonregular

- Pumping Lemma identifies nonregularity
  - Reg langs must satisfy its conditions

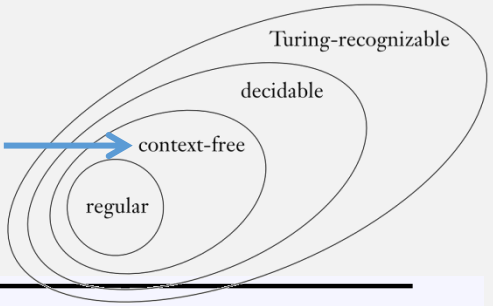


THEOREM 1.70

Pumping lemma If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1. for each $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

Chapter 2: Context-free Languages

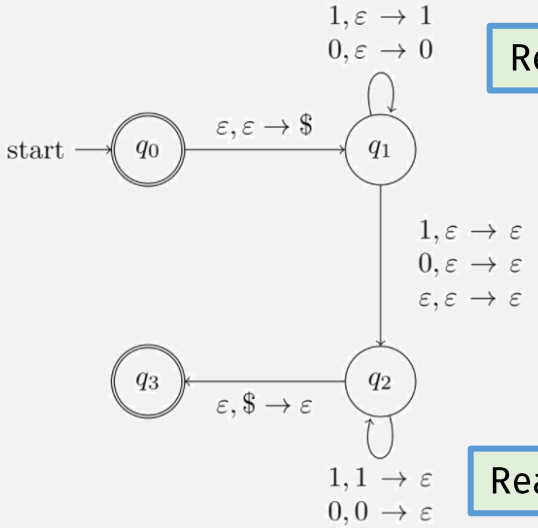


- Generated by a CFG
  - E.g., $L = \{w \mid w = \text{FLIP}(w)\}$

HW 5, Problem 2

  - S → ε | 0 | 1 | 0S0 | 1S1
- Recognized by a PDA
  - "NFA with a stack"
  - Enables pair dependency

HW 5, Problem 3



Read/push chars

Switch to 2nd half (nondet)

Read/pop matching chars

DEFINITION 2.2

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

DEFINITION 2.13

A ***pushdown automaton*** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where Q, Σ, Γ, and F are all finite sets, and

1. Q is the set of states,
2. Σ is the input alphabet,
3. Γ is the stack alphabet,
4. $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

The Limits of Context-free Languages

- CFLs can't have more than paired dependencies

- E.g. of non-CFL, $\{a^n b^n c^n | \, n \geq 0\}$
  - Also cannot match arbitrary strings, as in $\{ww | \, w \in \{0,1\}^*\}$
  - So XML is not a CFL! (but HTML is)

- CFL Pumping Lemma identifies non-CFLs

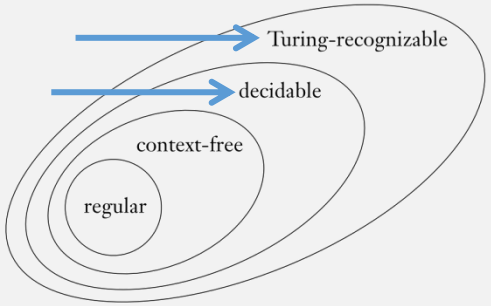
THEOREM 2.34

Pumping lemma for context-free languages If A is a context-free language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into five pieces $s = uvxyz$ satisfying the conditions

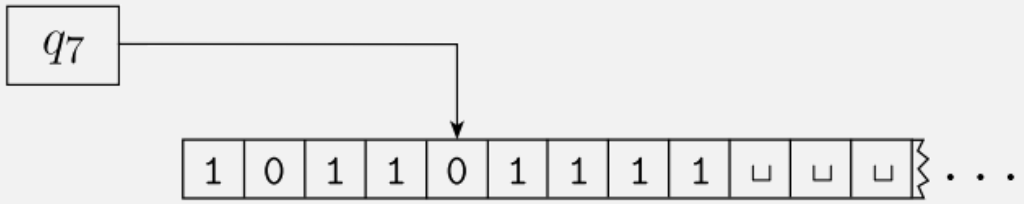
1. for each $i \geq 0$, $uv^i xy^i z \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$.

Paired dependency

Chapter 3: Turing Machines



- TMs have a tape, head can read/write and move L/R each step



DEFINITION 3.3

A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, 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. $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

- TMs can recognize langs with arbitrary dependencies

$$\{ww \mid w \in \{0,1\}^*\}$$

It's All About the Stacks

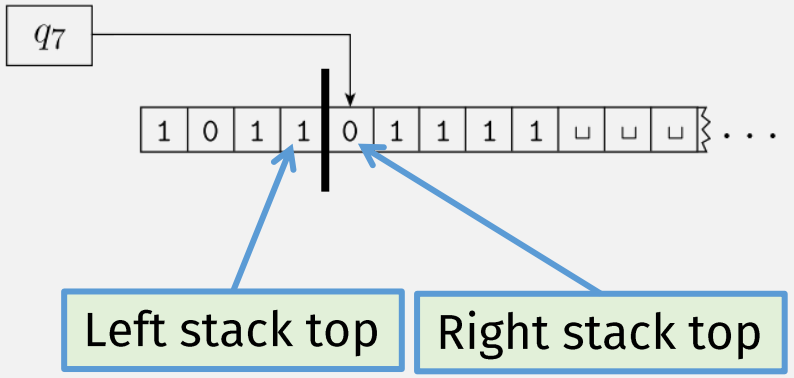
HW 6, Problem 4

An NFA has **no** stack. It recognizes regular languages.

A PDA is an NFA plus **one** stack. It recognizes context-free languages.

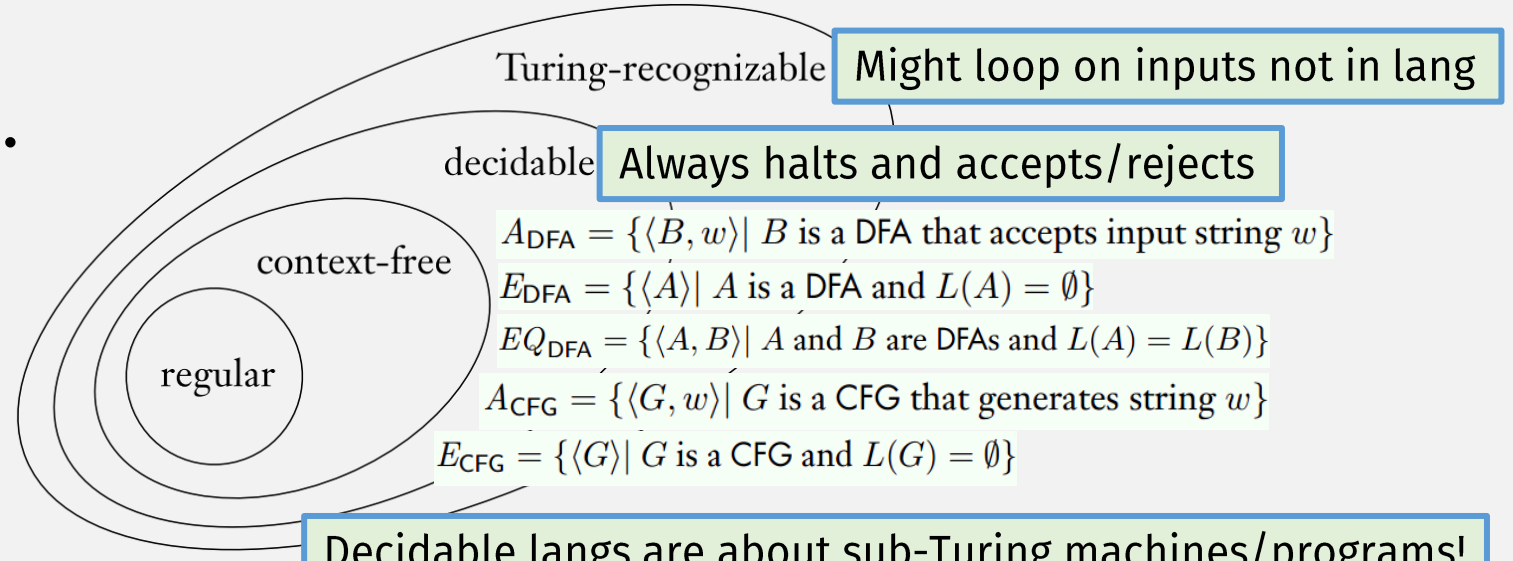
Prove that a PDA with **two** stacks recognizes Turing-recognizable languages.

- Proof setup:
  - Label stacks "left"/"right", representing TM tape to left/right of head, resp.
  - Formally, say TM head points to top of "right" stack
- TM start:
  - First, load input onto left stack (will be backwards)
  - Then, transfer to right stack (no longer backwards)
  - Now "head" is pointing to "start" of tape
- TM write:
  - Pop from right stack and push new char
- TM head move:
  - Left: Pop char from left stack and push onto right stack
  - Right: Pop char from right stack and push onto left stack



Chapter 4: TMs Are Too Powerful!

- They can loop!
  - This is undesirable for many programs
- So we further distinguish different TMs
- A lang is decidable if …
  - … it has a decider!



Determining Undecidability (the first lang)

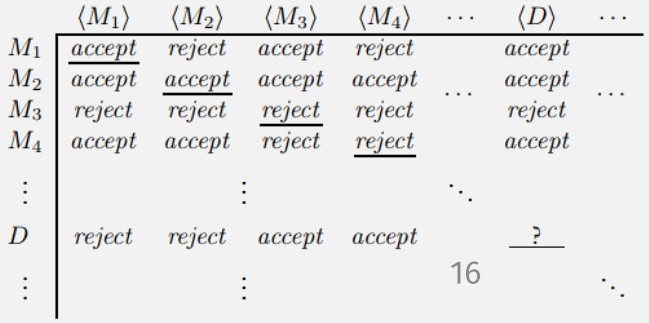
- A language is undecidable if … ??? (much harder)
  - (but not too hard)

- A_{TM} is undecidable, in 3 easy steps (proof by contradiction):
  1. Assume A_{TM} is decidable and has decider S
  2. Create another decider D that, on input <M>:
     - Runs S on <M,<M>> and does the opposite
  3. Run D with itself as input <D>:
     - If D accepts itself, then it must reject
     - If D rejects itself, then it must accept

$$A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$$

CONTRADICTION

Creating this contradiction requires
(Cantor's) diagonalization
(An imporant result in CS!)



Undecidability and Mapping Reducibility

- Easier to prove undecidability via reduction

THEOREM 5.22

If $A \leq_m B$ and B is decidable, then A is decidable.

COROLLARY 5.23

If $A \leq_m B$ and A is undecidable, then B is undecidable.

DEFINITION 5.17

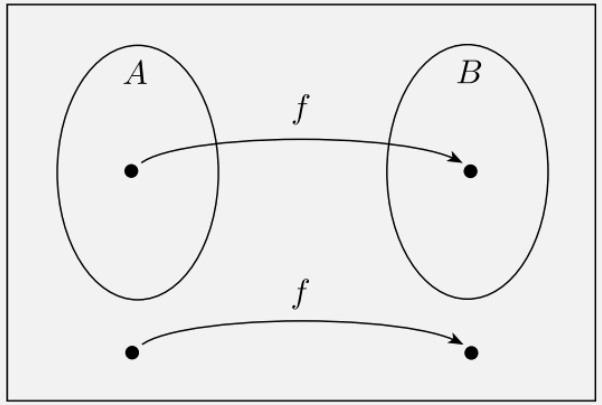
A function $f: \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

DEFINITION 5.20

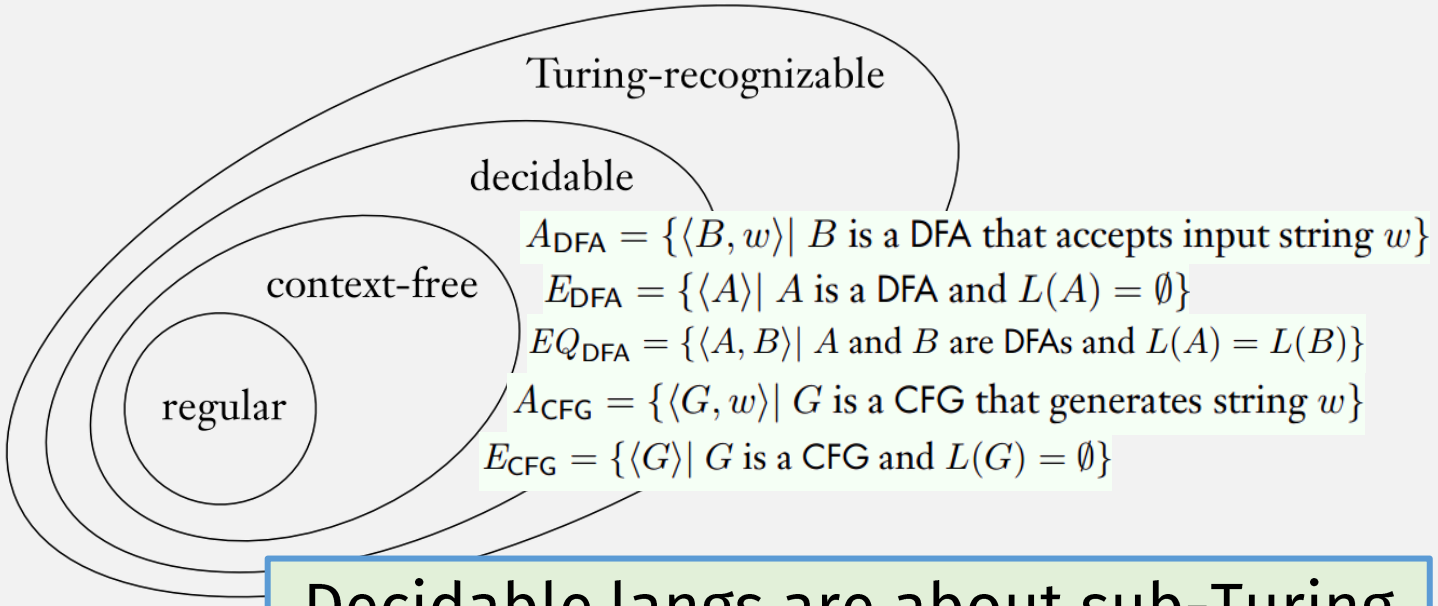
Language A is ***mapping reducible*** to language B, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \longrightarrow \Sigma^*$, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***reduction*** from A to B.



Previously: Decidable Langs, Sub-Turing progs



Decidable langs are about sub-Turing machines/programs!

Some **Un**decidable Langs

$$E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

$$REGULAR_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}$$

$$\text{CONTEXTFREE}_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a CFL}\}$$

HW 9, Problem 2

$$\text{ACCEPTEVERYTHING}_{TM} = \{\langle M \rangle \mid M \text{ is a TM that accepts any string in } \Sigma^*\}$$

HW 8, Problem 2

Undecidable Langs Follow the Same Pattern!

$$E_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$$

$$REGULAR_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a regular language}\}$$

$$\text{CONTEXTFREE}_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a CFL}\}$$

$$\text{ACCEPTEVERYTHING}_{TM} = \{\langle M \rangle \mid M \text{ is a TM that accepts any string in } \Sigma^*\}$$

- Rice's Theorem: HW 9, Problem 4
  - All langs of the following form are undecidable:

$$\{\langle M \rangle \mid \ldots M \text{ is a TM and } \ldots \text{ something about } L(M) \ldots\}$$

Rice's Theorem

This is called *Rice's Theorem*, which states that all languages P that satisfy the following conditions are undecidable:

- P is not the empty set;
- P consists of only valid Turing machine descriptions;
- P is not the set of all Turing machine descriptions;
- and whether a Turing machine is in P is determined by examining the language of that Turing machine, i.e., P has the form:

$$\{\langle M \rangle \mid \ldots M \text{ is a TM and } \ldots \text{ something about } L(M) \ldots\}$$

HW 9, Problem 4

Standard undecidability proof (by contradiction) using A_{TM} :

- Assume a language P satisfying Rice's Theorem is decided with TM R:
  - So there is some TM M_{P} in P accepted by R
- Use R to construct TM deciding A_{TM}:
  - On input <M, w>:
    - Construct M' that on input x:
      - If M accepts w, run M_{P} on x and accept if accept
      - Else reject
    - Run R with input <M'>, accept if accept, reject if reject

R accepts TMs in P, e.g., M_P

so $M' = M_{\text{P}}$ if M accepts w

But A_{TM} doesn't have a decider, so we have a contradiction!

More Undecidable Languages

- Rice's Theorem:
  - All langs of the following form are undecidable

$$\{\langle M \rangle \mid \ldots M \text{ is a TM and } \ldots \text{ something about } L(M) \ldots\}$$

HW 9, Problem 5

All are undecidable, since they satisfy the conditions of Rice's Theorem!

Prove that the following languages are undecidable:

1. $L_1 = \{\langle M \rangle \mid M \text{ is a TM and } \texttt{"CS420 rules!"} \in L(M)\}$
2. $L_2 = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is the set of all Rust programs}\}$
3. $L_3 = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ consists of C++ malware that installs bitcoin miners on your computer}\}$

5 More Undecidable Languages

1. $L_1 = \{\langle M \rangle \mid M \text{ is a TM and } \texttt{"CS420 rul}$
2. $L_2 = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is the}$
3. $L_3 = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ consi}$

https://www.cs.umb.edu/~stchang/cs420/s21/hw9.html#%28part._hw9-ricelangs%29

The Limits of Turing Machines

- A_{TM} is at least recognizable

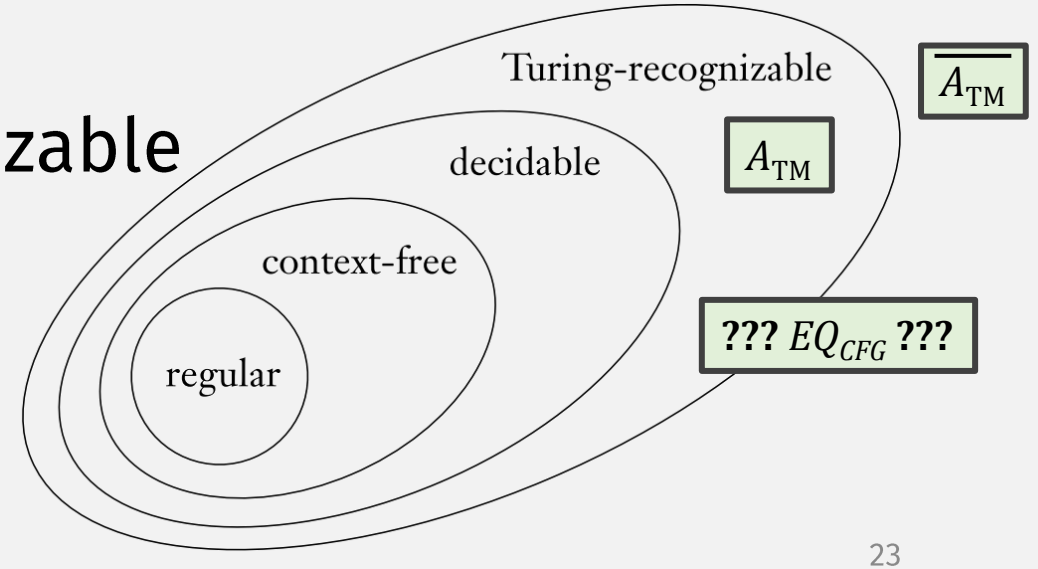
U = "On input $\langle M, w \rangle$, where M is a TM and w is a string:

1. Simulate M on input w.
2. If M ever enters its accept state, *accept*; if M ever enters its reject state, *reject*."

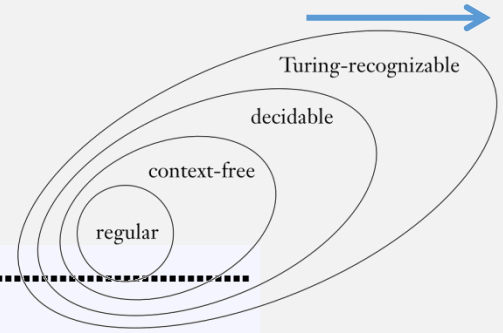
- Some languages aren't even recognizable

HW 8, Problem 5

- We know EQ_{CFG} is undecidable
  - Using proof by contradiction method for proving undecidability



Unrecognizability



- Key Theorem

THEOREM 4.22

A language is decidable iff it is Turing-recognizable and co-Turing-recognizable.

- Example,

Show that EQ_{CFG} is not recognizable either. HW 8, Problem 5

You may assume that the theorem from EQCFG is not Decidable has already been proved.

Hint: Show that EQ_{CFG} is co-Turing-recognizable first.

- EQ_{CFG} is undecidable, so to prove unrecognizability … (via thm 4.22)
  - … must show co-recognizability
  - i.e., $\overline{EQ}_{\text{CFG}}$, the language of unequal grammars, is recognizable
- Recognizer for $\overline{EQ}_{CFG}$: On input <G_1,G_2>:
  - For i = 1, 2, …
    - Run A_{CFG}'s decider with both <G_1,s_i> and <G_2,s_i>
      - where s_1, … is all possible strings
    - if one is accepted and one is not, then we know the langs are different so accept
    - otherwise keep trying strings
      - if the langs are different, will eventually find a string that differentiates the grammars
      - Otherwise it will loop forever, which is ok for a recognizer

$A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$

Set of all strings is countable, so it's possible to order them

Chapter 6: The Recursion Theorem

TM descriptions can "get a copy of themselves"

- Alternate proof of Rice's Theorem (by contradiction):
- Assume some lang P satisfying Rice's Theorem has decider R:
  - So there is some M_{P} in P accepted by R
  - And some M_{notP} not in P rejected by R

R accepts TMs in P

Construct the following impossible TM:

- S = On input x:
  - Obtain own description <S> and run R on <S>
  - If accept, run M_{notP} on x
  - If reject, run M_{P} on x

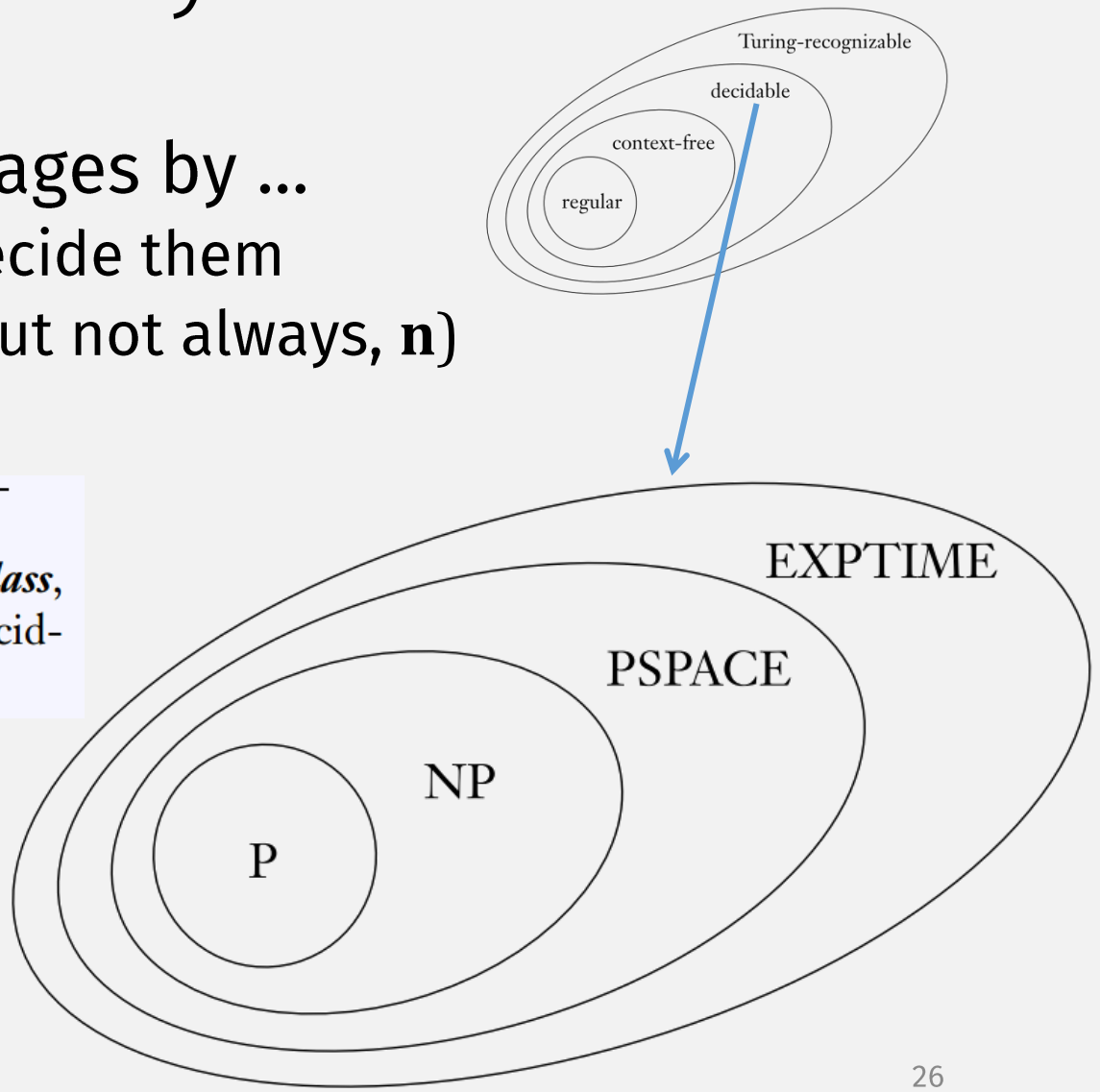
- If R accepts S, then S in P, but $S = M_{\mathrm{notP}}$, which is not in P
- If R rejects S, then S not in P, but $S = M_{\mathrm{P}}$, which is in P

Chapter 7: Time Complexity

- Further classify (decidable) languages by …
  - … how many steps are needed to decide them
  - Depends on size of input (usually, but not always, **n**)
  - Big-O = "as **n** gets large"

DEFINITION 7.7

Let $t: \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. Define the ***time complexity class***, **TIME($t(n)$)**, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.



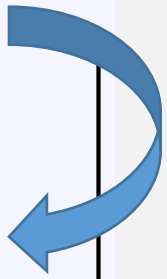
Big-O

DEFINITION 7.2

Let f and g be functions $f, g: \mathcal{N} \longrightarrow \mathcal{R}^+$. Say that $\boldsymbol{f(n) = O(g(n))}$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$,

$$f(n) \leq c\, g(n).$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.



TRUE	1. $4n + 2 = O(n)$
FALSE	2. $n^2 + 4n + 2 = O(n)$
TRUE	3. $n^2 + 4n + 2 = O(n^2)$
FALSE	4. $n^2 + 4n + 2 = O(n \log n)$
TRUE	5. $n^2 + 4n + 2 = 2^{O(n)}$
TRUE	6. $100^{11n} + 5 = 2^{O(n)}$

HW 10, Problem 5

HW 10, Problem 4

Running Time of $\text{ACCEPTEVERYTHING}_{DFA}$

$$\text{ACCEPTEVERYTHING}_{DFA} = \{\langle D \rangle \mid D \text{ is a DFA that accepts any string in } \Sigma^*\}$$

Show that the $\text{ACCEPTEVERYTHING}_{DFA}$ language (from the DFAs That Accept Everything problem in Homework 8) is in P.

If it would help, here's a decider M for $\text{ACCEPTEVERYTHING}_{DFA}$:

M = on input $\langle D \rangle$, where D is a DFA:

1. Apply S to $\langle D \rangle$ to get $\langle D_{not} \rangle$.
2. Apply T to $\langle D_{not} \rangle$ and accept if T accepts, else reject.

where:

- S is a machine implementing a solution for the The Complement Operation for Regular Languages (Bonus) problem from Homework 7. **easy**

  Specifically, S, on input $\langle D \rangle$, where D is a DFA $\langle Q, \Sigma, \delta, q_0, F \rangle$, changes its tape contents to $\langle Q, \Sigma, \delta, q_0, Q - F \rangle$.

  In other words, accept states in D become non-accept states in the output, and vice versa. All other parts of the DFA are kept the same.

- T is the decider for E_{DFA} from Theorem 4.4. **harder**

Run Time of E_{DFA}

$$E_{\text{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$$

- Let **q** = # states

T = "On input $\langle A \rangle$, where A is a DFA:

1. Mark the start state of A. O(**1**)
2. Repeat until no new states get marked: Repeat at most **q** times = O(**q**)
   - Mark any state that has a transition coming i
   - state that is already marked
4. If no accept state is marked, *accept*; otherwise, *reject*." O(**q**)

Max transitions per state = **q**

Max marked states = **q**

Total: # loops * steps per loop = O(**q**) * O($\mathbf{q^2}$) = O($\mathbf{q^3}$)

- Total = O(**1**) + O($\mathbf{q^3}$) + O(**q**) = O($\mathbf{q^3}$)

Chapter 7: Time Complexity

- Further classify (decidable) languages by …
  - … how many steps are needed to decide them
  - Depends on size of input **n**
  - Big-O = "as **n** gets large"

DEFINITION 7.7

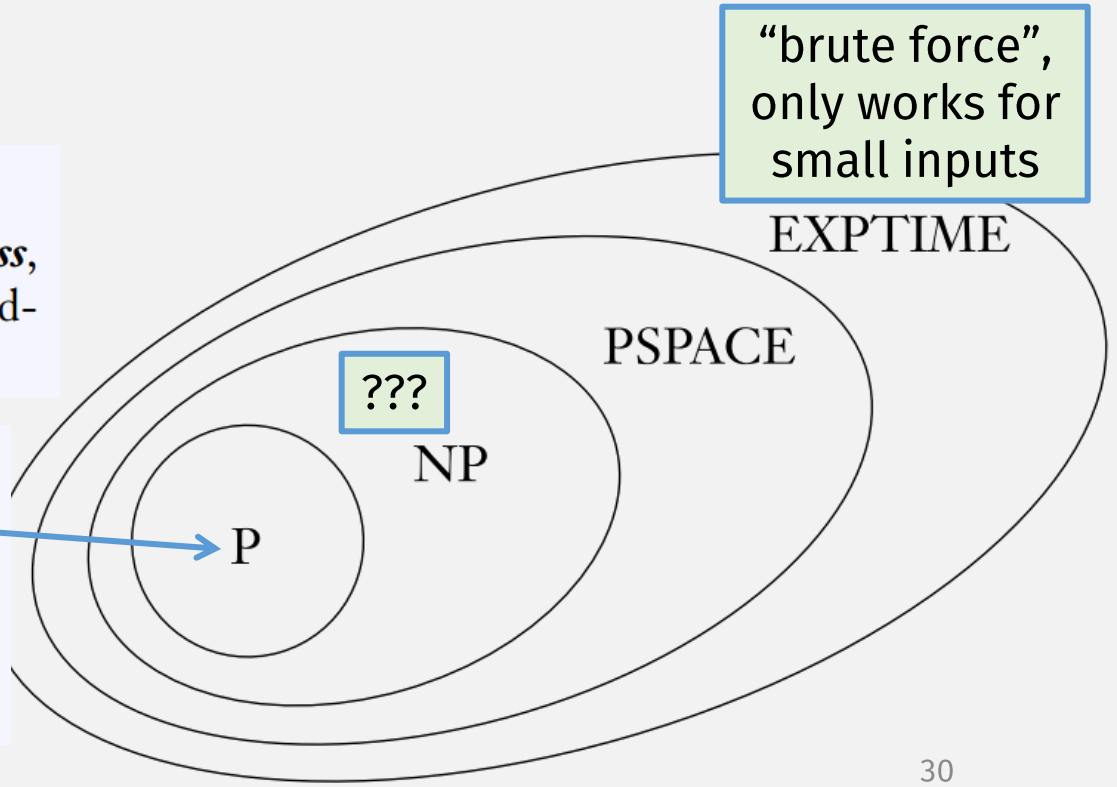
Let $t: \mathcal{N} \longrightarrow \mathcal{R}^+$ be a function. Define the ***time complexity class***, **TIME($t(n)$)**, to be the collection of all languages that are decidable by an $O(t(n))$ time Turing machine.

DEFINITION 7.12

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$\mathrm{P} = \bigcup_k \mathrm{TIME}(n^k).$$

"solvable" in practice



Verifying Strings in a Language

- Brute force: check *every* possible solution
- Verifier: check *one* possible solution

DEFINITION 7.18

A ***verifier*** for a language A is an algorithm V, where

$$A = \{w | \ V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}.$$

Certificate or "proof"

We measure the time of a verifier only in terms of the length of w, so a ***polynomial time verifier*** runs in polynomial time in the length of w. A language A is ***polynomially verifiable*** if it has a polynomial time verifier.

DEFINITION 7.19

NP is the class of languages that have polynomial time verifiers.

P vs NP

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$$

$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

Problem	Brute Force Solution?	Poly Time Solution?	Poly Time Verifier?
PATH	Yes	Yes	Yes
HAMPATH	Yes		Yes

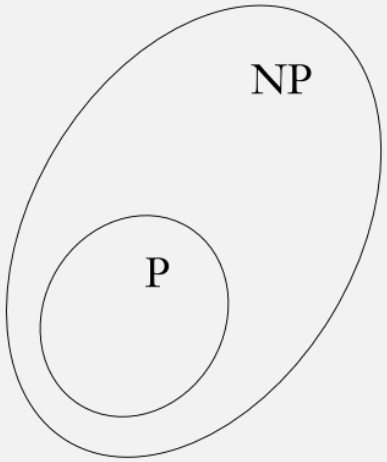
P = NP?

$$PATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t\}$$

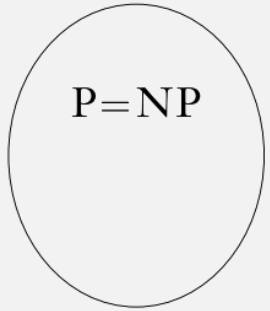
$$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$$

Problem	Brute Force Solution?	Poly Time Solution?	Poly Time Verifier?
PATH	Yes	Yes	Yes
HAMPATH	Yes	Maybe?	Yes

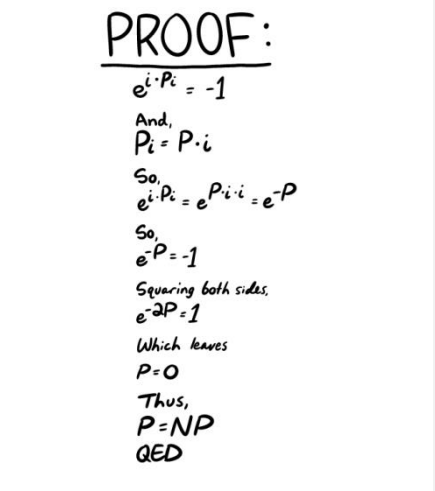
The biggest unsolved mystery in computer science



?OR?



Implication if P=NP:
all unsolvable problems become solvable!



NP-Completeness

DEFINITION 7.34

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

THEOREM 7.35

If B is NP-complete and $B \in \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$.

We need to know one **NP**-Complete problem in order to prove others!

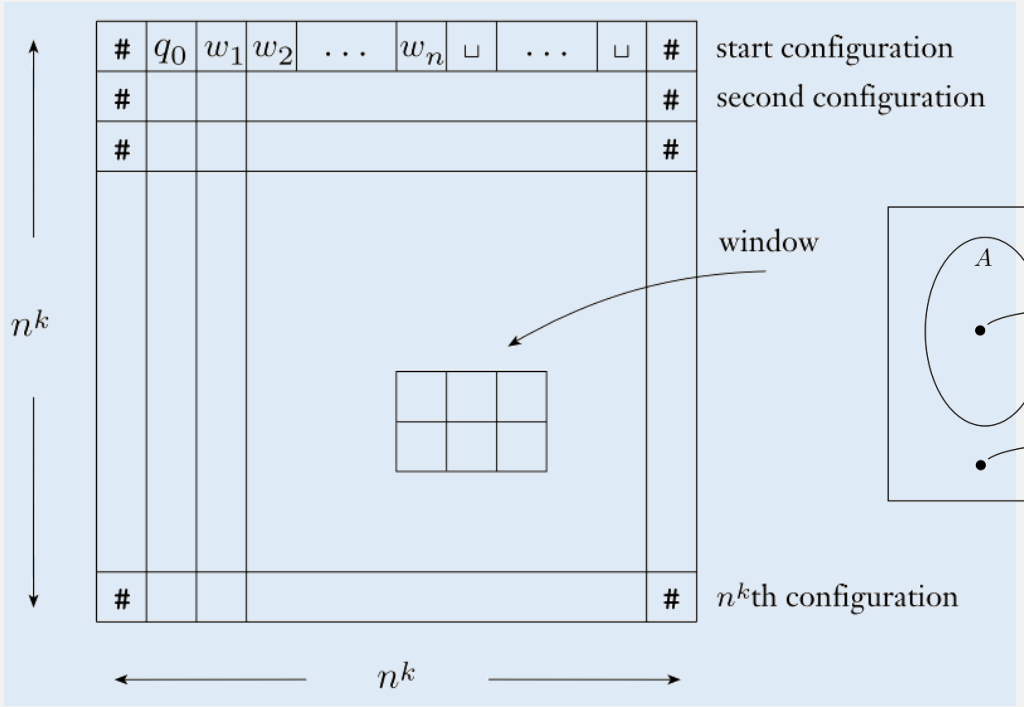
The First NP-Complete Problem

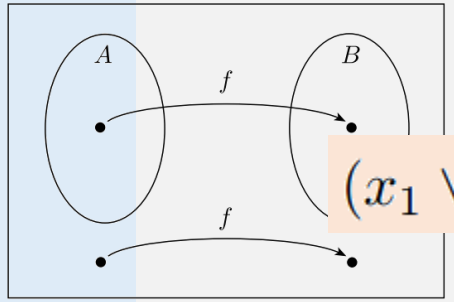
THE COOK–LEVIN THEOREM

THEOREM 7.37

SAT is NP-complete

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$





$$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6})$$

More NP-Complete Problems

THEOREM 7.36

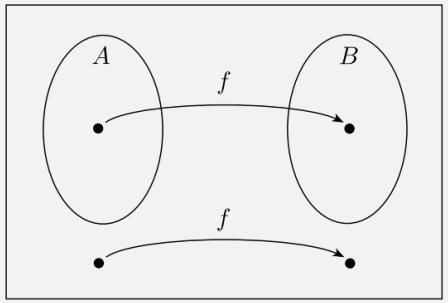
If B is NP-complete and $B \leq_{\mathrm{P}} C$ for C in NP, then C is NP-complete.

DEFINITION 7.29

Language A is ***polynomial time mapping reducible***,[1] or simply ***polynomial time reducible***, to language B, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f : \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \iff f(w) \in B.$$

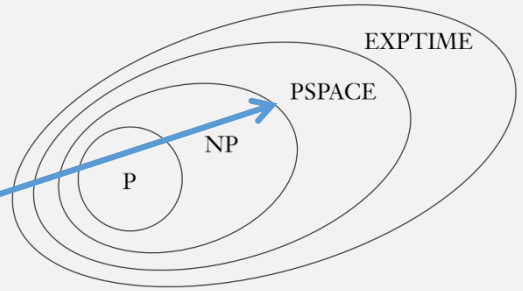
The function f is called the ***polynomial time reduction*** of A to B.



NP-Complete problems, so far

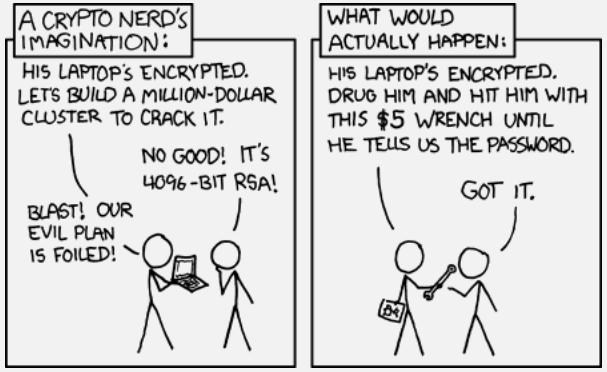
- $SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$ (Cook-Levin Theorem)
- $3SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$ (reduce from SAT)
- $CLIQUE = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$ (reduce from $3SAT$)
- $HAMPATH = \{\langle G, s, t \rangle \mid G \text{ is a directed graph with a Ham. path from } s \text{ to } t\}$
  - (reduce from $3SAT$)
- $UHAMPATH = \{\langle G, s, t \rangle \mid G \text{ is a undirected graph with a Ham. path from } s \text{ to } t\}$
  - (reduce from $HAMPATH$)
- $SUBSET\text{-}SUM = \{\langle S, t \rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\}$ (reduce from $3SAT$)
- $VERTEX\text{-}COVER = \{\langle G, k \rangle \mid G \text{ is an undirected graph that has a } k\text{-node vertex cover}\}$ (reduce from $3SAT$)

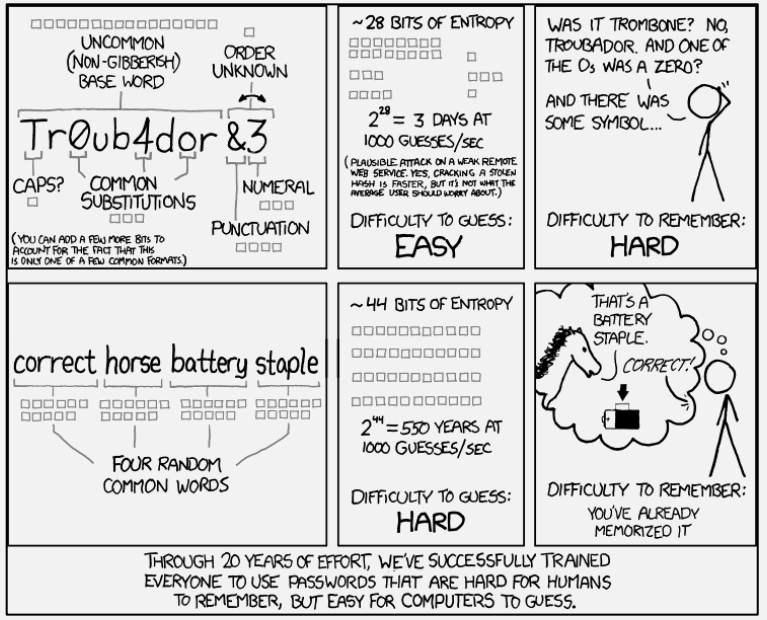
Beyond CS420



Thank you!

- Space complexity
  - Dynamic programming improves run time by using more space
  - Space-time tradeoff
- Other decidable "logics"
  - E.g., can a computer prove statements like x + 0 = x
- Information theory
- Cryptography
- Probablistic Algorithms and "Randomness"






```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```