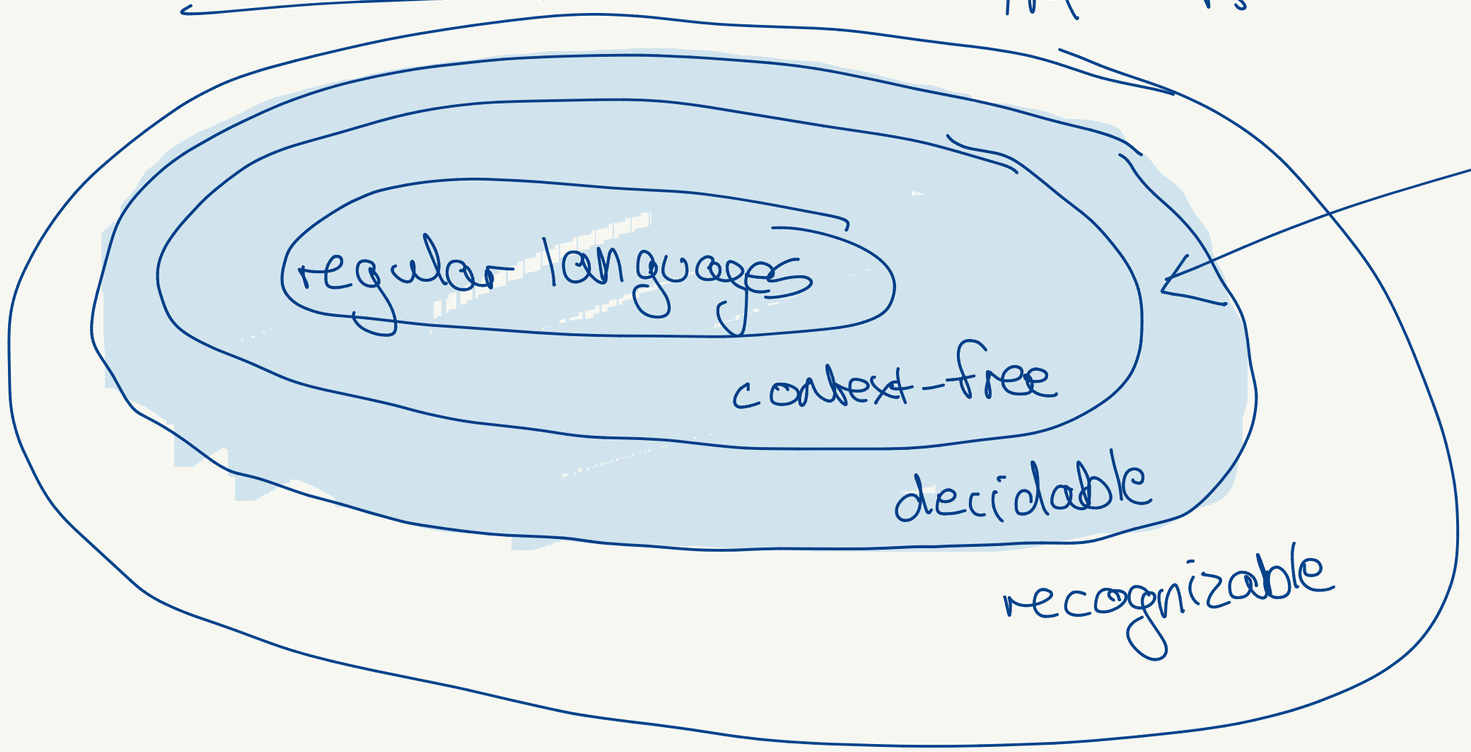
FOCS Lecture 26: Efficiency / Tractability

- Time Complexity
- P: class of efficiently solvable problems
  - Extended Church-Turing Thesis
- A decidable problem that is not in P
- On the boundary between efficiently solvable problems and those that are not (NP)

- Recap of what we've learned on computation

Last Time: Decidability and Unsolvable Problems

- Universal Turing Machine U_{TM}, simulates action of turing machine M on input w, given input $\langle M \rangle \# w$
- $\mathcal{L}_{TM} = \{\langle M \rangle \# w : M \text{ accepts } w\}$ is undecidable
- $\mathcal{L}_{HALT} = \{\langle M \rangle \# w : M \text{ halts given input } w\}$ is also undecidable, because $\mathcal{L}_{TM} \leq_R \mathcal{L}_{HALT}$



we work here 99% of the time as practical computer scientists

unrecognizable languages, e.g. $\overline{\mathcal{L}_{TM}}$

Today: Efficiently solvable problems

Ex: $\mathcal{L}_{prime} = \{ w \mid w \text{ is prime} \}$ is this efficiently solvable?

Solvable, via trial division:

```
prime(w):
    for f ∈ [2, w-1]:
        if w % f == 0:
            return REJECT
    return ACCEPT
```

Natural questions

- This is an algorithm, ~~and can be implemented on a TM~~, but is it efficient?
- Is there any algorithm that solves this and is efficient?

How do we define efficiency

- Usually: quick, using little memory

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

M = Turing Machine that solves L

Input: Binary string w

1. Check that input has correct format and return to *
2. Match each 0 (left of #) to a 1 (right of #)
3. If a match fails or there are more 1s left, REJECT
   Else ACCEPT

ACCEPT

Lower-level description

M

Input: Binary string w

1. Check that input has correct format and return to *
2. Match each 0 (left of #) to a 1 (right of #)
   - Move right and mark the first unmarked 0 (if none, GOTO step 3)
   - Move right and mark the first unmarked 1 (if none, REJECT)
   - Move left to the first unmarked 0
3. If there are any unmarked 1s, REJECT. Otherwise ACCEPT

NOW we can analyze the runtime of M (number of moves of the machine head to ACCEPT or REJECT)

runtime of M depends on two factors:

- the "size" of the input. E.g. here we must check longer strings by using more moves of the machine head.
- even for fixed size inputs, the runtime depends on the specific input. E.g. if w starts with a 1, M will reject w in step 1 immediately.

Thus we look at <u>worst case</u> runtimes for <u>fixed input sizes</u>

To get the runtime of M:

1. identify a parameter defining the size (or complexity) of the input
2. Fix that size n and identify the worst input w^* (of size n)
3. Determine the number of moves of the machine head for that input w^*

results in runtime

$T_M(n)$ ← a function of the input size

We only care about the asymptotic behavior ("growth rate")
of T_M with respect to input size n.

E.g. $T_M = O(f)$ means $T_M(n) \lesssim f(n)$

$T_M = \Theta(f)$ means $T_M(n) \approx f(n)$

We ignore constants because:

1. Tracking them accurately requires a tedious low-level (machine level) description of the algorithm
2. They change with even slight changes of the machinery
3. They don't matter (if small) as much as the growth rate does

Worst-case runtime analysis for M

- take our parameter defining size/complexity for

$$\mathcal{L} = \{0^n \# 1^n \mid n \geq 0\}$$ to be n

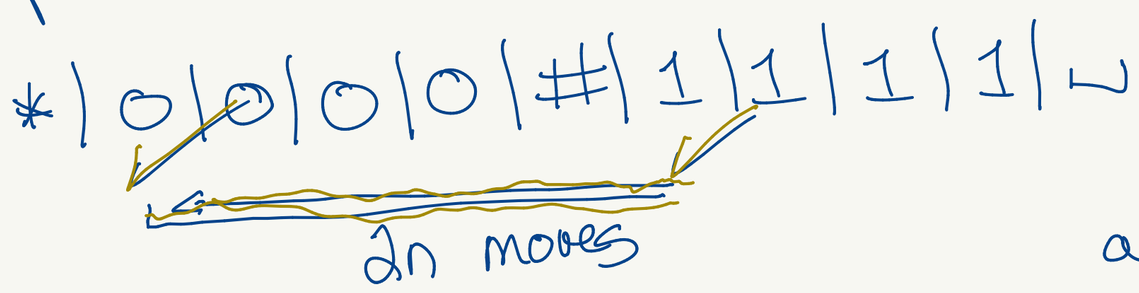
- notice that the worst case input of size n is

$$w^* = 0^n \# 1^n$$

any other string with this complexity will be rejected in step 1

Step 1 of M: check input has correct format, can be done in $\Theta(n)$ steps

Step 2 of M: match each 0 (left of #) to a 1 (right of #)



it requires about $2n$ steps to match each 0-1 pair and there are n such 0-1 pairs so it takes about $2n^2$ steps to complete Step 2

$\Rightarrow T_M(n) = \Theta(n^2)$

we say M is a quadratic-time algorithm for L.

Q: is there a more efficient algorithm for this problem?

How could we get one:

- be more clever! see book: use a halving trick to get an $\Theta(n \log n)$ algorithm

- change the architecture! e.g. use a machine w/ two-tapes and two independent machine heads

input

tape 1 ␣ * 0 0 0 0 # 1 1 1 1 → copy $\Theta(n)$ → * 0 0 0 0 # 1 1 1 1

tape 2 ␣ * ␣ ␣ ...

* 0 0 0 0 ␣ ...

$\Rightarrow \Theta(n)$ algorithm

move tape 1 head right & move tape 2 head left, matching as they go $\Theta(n)$

Important Lesson

- In terms of solvability, all reasonable TM architectures ($\Sigma = \{0, 1, \#, *\}$) (say multi-tape or with more symbols in the input set) have the same power as single-tape TMs, as both can be simulated using U_{TM}, a single-tape TM. See chap 27
- But in terms of ~~efficiency~~ runtime, they can differ greatly

IRL: desktop computers $\ll$ powerful than supercomputers

P: The Class of Efficiently Solvable Problems

We want a definition that

- defines efficiency independent of the specific TM architecture used
- does not depend on the efficiency of any specific algorithm that solves the problem
- gives a clear divide between tractable & intractable

This leads to $\mathcal{P}$ = the class of polynomially solvable problems

$L \in \mathcal{P}$ iff there exists an algorithm (decider) M that solves (decides) L and satisfies $T_M(n) = O(n^k)$ for some constant k

Things to note:

- $\mathcal{L} \in P$ if there <u>exists</u> an M that runs in worst-case polynomial time; there may of course be inefficient algorithms

- Whether $\mathcal{L} \in P$ depends on how the input size is parametrized & the computational model

Ex trial division for L_{prime}

for $f \in \{2, \omega-1\}$:

If $\omega \% f = 0$

REJECT

ACCEPT

Model 1

Consider $\omega \in N$ to have size ω, then

$$T_M(\omega) = \sum_{i=2}^{\omega-1} 1 = O(\omega)$$

linear time alg

Model 2

Consider $\omega \in \{0,1\}^n$ a binary string of length n, then the worst case input of size n is 2^{n-1}, then

$$T_M(n) = O(2^{n-1})$$

so exponential time

Extended Church–Turing Thesis (Thm)

If a problem is solvable in polynomial time on any "reasonable" Turing Machine architecture, then it is solvable in polynomial time on a one-tape Turing Machine, and vice versa.

= "reasonable"

$\Sigma = \{0, 1, \#, *\}$

The class P is independent of the ^ Turing Machine architecture used to measure the (asymptotic) runtime.

Decidable but non-efficient problems exist

- To show $\mathcal{L} \notin \mathcal{P}$, we have to show that any algorithm solving it has superpolynomial runtime

$|\omega|$ = length(ω)

Consider

$$\mathcal{L}_{TM\text{-}EXP} = \{ \langle M \rangle \# \omega \mid M \text{ accepts } \omega \text{ within at most } 2^{|\omega|} \text{ steps} \}$$

Note that the boundedness of the runtime here is what makes $\mathcal{L}_{TM\text{-}EXP}$ decidable even though $\mathcal{L}_{TM}$ is undecidable:
we simply run $U_{TM}(\langle M \rangle \# \omega)$ and keep track of the # of steps of M that are simulated, and REJECT if more than $2^{|\omega|}$ steps are used.

Thm $\mathcal{L}_{TM\text{-}EXP} \notin P$: there is no polynomial time decider for $\mathcal{L}_{TM\text{-}EXP}$

Pf (sketch)

By contradiction. Assume A decides $\mathcal{L}_{TM\text{-}EXP}$ and has runtime $\leq Cn^k$ where $n = |\langle M \rangle \# w|$ is the size of the input string.

Construct D a decider such that

$$D(\langle M \rangle) = \begin{cases} \text{ACCEPT if } A(\langle D \rangle \# \langle M \rangle) \text{ is REJECT} \\ \text{REJECT if } A(\langle M \rangle \# \langle M \rangle) \text{ is ACCEPT} \end{cases}$$

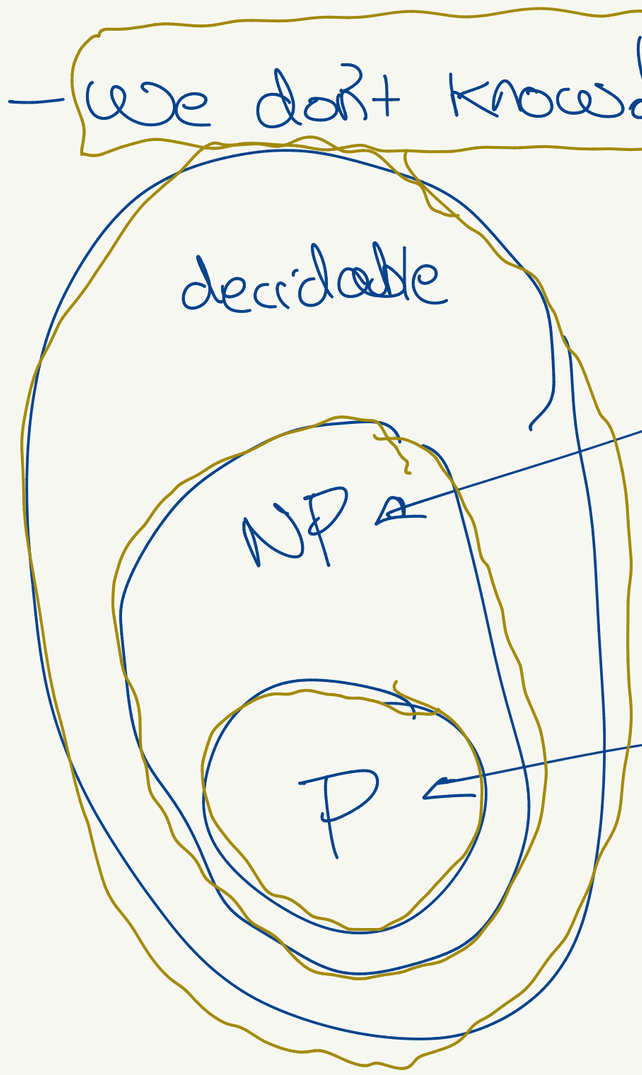
Show that D neither accepts nor rejects itself. This is a contradiction.

Conclude that D does not exist, therefore A does not exist.

$2^{|\langle M \rangle|}$

∎

Question: Do there exist practically interesting problems that are not in P?



problems whose solns can be verified efficiently: e.g. is a given tour of a graph a Hamiltonian tour?

problems whose solns can be computed efficiently

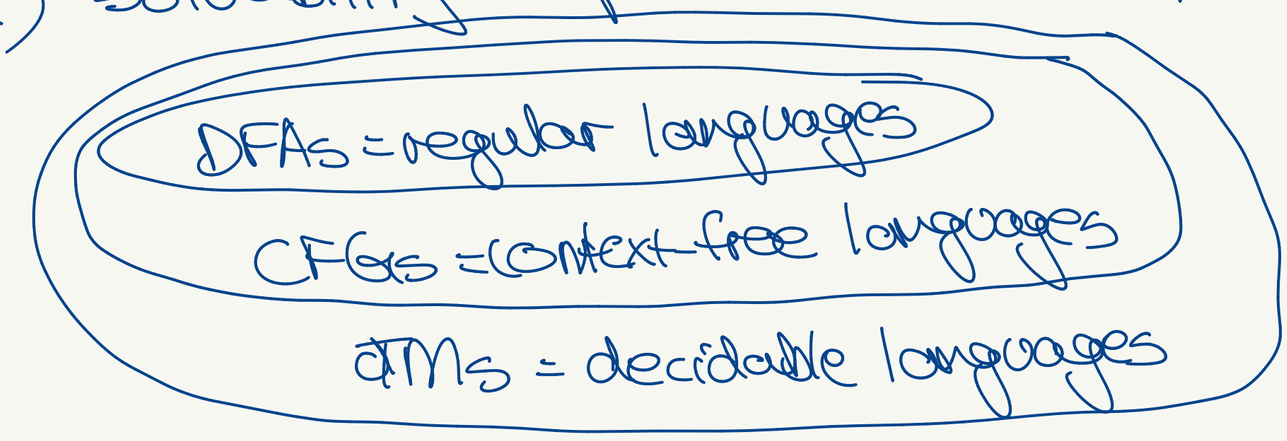
$P \subsetneq NP$?

$P \neq NP$?

Recap of Computation Theory

1) problems = decision problems = languages

2) solvability depends on the computational model



3) algorithms = TM deciders

4) (Church-Turing Thesis) a problem is solvable with an algorithm iff there is a TM decider for that language

5) tractable problems = P = set of languages for which deciders w/ polynomial worst case runtime exist

That's IT! (for FOCS)