Announcements:

- HW4 due today @ 11:59pm
- HW5 due next Tues @ midnight
- Final Exam:
  - Available on Gradescope from 12:01 AM on Weds 6/29 - 9:00pm on Friday 7/1
  - Can work for any consecutive 12 hours during this block.
  - Open resources: book, notes, class resources

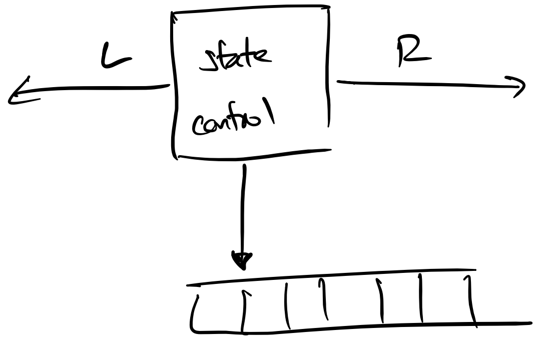
    Closed resources - peers, internet (except unrelated LaTeX)
- Posting on Ed: clarification only
- Course Assessments - CW/Ed.

---

1. TMs and their languages (cont'd)

2. Variant TMs.
   - 2.1) Multitape
   - 2.2) Nondeterministic
   - 2.3) Enumerators

3. Undecidable language, unrecognizable language.



- read
- change int. state
- write
- move

Recognizable: L is recognizable if ~~or~~ some TM halt and accept on exactly the strings in the language

Decidable: L is decidable if some TM halts, accepts all strings in the language; halts, rejects on all strings not in the language.

Decidable:

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

$B = \{w\#w \mid w \in \{0,1\}^*\}$

$C = \{a^i b^j c^k \mid i \times j = k\}$

$D = \{\#x_1 \# x_2 \cdots \# x_\ell \mid x_i \neq x_j \text{ if } i \neq j\}$

$A_{DFA} = \{\langle D, w\rangle \mid D \text{ a DFA that accepts } w\}$

Recognizable:

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

Church-Turing thesis

"what computers can do"
"our intuitive notion of algorithm" $\approx$ "what TMs can do."

Decide

- $A_{NFA} = \{\langle N, w\rangle \mid N \text{ is an NFA that accepts on } w\}$.
  - simulate $N(w)$!
  - $N \Rightarrow$ DFA, use our TM for A_{DFA}.

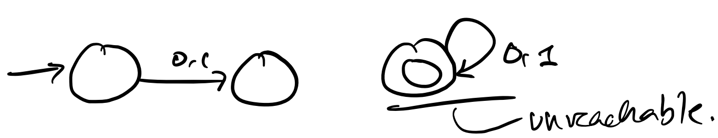
- A_{REX} : similar.
  $= \{\langle R, w\rangle \mid R \text{ is a reg. ex and } R \text{ matches } w\}$.

Tool 1: using previous deciders as subroutines.

- $E_{DFA} = \{\langle D\rangle \mid D \text{ is a DFA that accepts no strings.}\}$

To decide:

1. Mark the start state
2. BFS on the state diagram to find all reachable states.
3. Accept if and only if no reachable accept states.

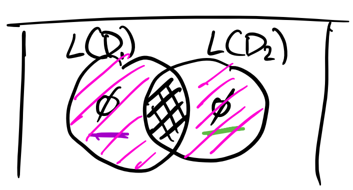


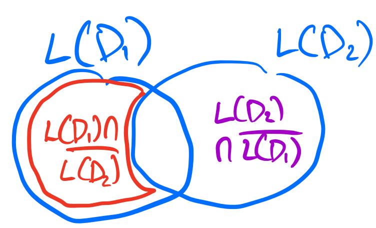
Decide

$EQ_{DFA} = \{\langle D_1, D_2\rangle \mid D_1, D_2 \text{ are DFAs and } L(D_1) = L(D_2)\}$

"simulate D_1 and D_2 on all strings"

To decide: use regular operations.





Our decider will do the following:

1) Build DFAs for $\overline{L(D_1)}$ and $\overline{L(D_2)}$
(swap accept, reject).

2) Using our procedure that builds DFAs for $A \cup B$, $A \cap B$,
Build DFA for:

$$\left(\overline{L(D_1)} \cap L(D_2)\right) \cup \left(L(D_1) \cap \overline{L(D_2)}\right)$$

3) Simulate E_{DFA} on this new DFA and accept
if and only if E_{DFA} accepts.

$$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG } L(G) = \emptyset\}$$

$S \to SS$
$A \to 000$
no example.

$S \to AA \mid A0A$
$A \to 11 \mid 00 \mid B$
$B \to S$

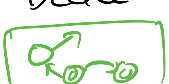
• → can get to terminals from here.

Algorithm for deciding.

1. Mark all terminals with a dot.
2. While the marked set is still increasing —
   - mark all variables that produce a string of only marked symbols with a dot.
3. Accept if and only if start state is not marked.

Puzzles.

(1) Decide $ALL_{DFA} = \{\langle D \rangle \mid D \text{ is a DFA that accepts all strings}\}$.



– Start at the $\to$O, we BFS to check if all reachable states are accepts ◎.

(2) Decide $A_{CFG} = \{\langle G, w \rangle \mid G \text{ is a CFG that generates } w\}$

– Convert G to a PDA and simulating.

(+) Recognize $\overline{E_{TM}} = \{\langle M\rangle \mid M$ is a TM that accepts at least one string$\}$

- BFS over internal states $Q(M)$?

(+) Given a CFG G for a language A, design a TM that recognizes A.

Idea 1: Simulate M on $\varepsilon, 0, 1, 00, 01 \ldots$ until M accepts on some string.

$\uparrow$ 4 steps

Idea 2:

strategy for dodging loops!

For $i = 0, 1, 2, \ldots$
  If $s_0, s_1, \ldots$ is a list of all strings in Σ^*
  Simulate M on strings s_0 through s_i for i steps each, or until halts.
Accept if any simulation accepts.

(Suppose $s_{1000} \in L(M)$, and $M(s_{1000})$ accepts in t steps. our TM finds out on the iteration of the loop $\max(1000, t)$.)

(+) Given a CFG G for a language A, design a TM that recognizes A.

Know: There's some TM (call it S) that decides
$$A_{CFG} = \{\langle G, w\rangle \mid G \text{ generates } w\}.$$

Our TM that decides A works as follows:

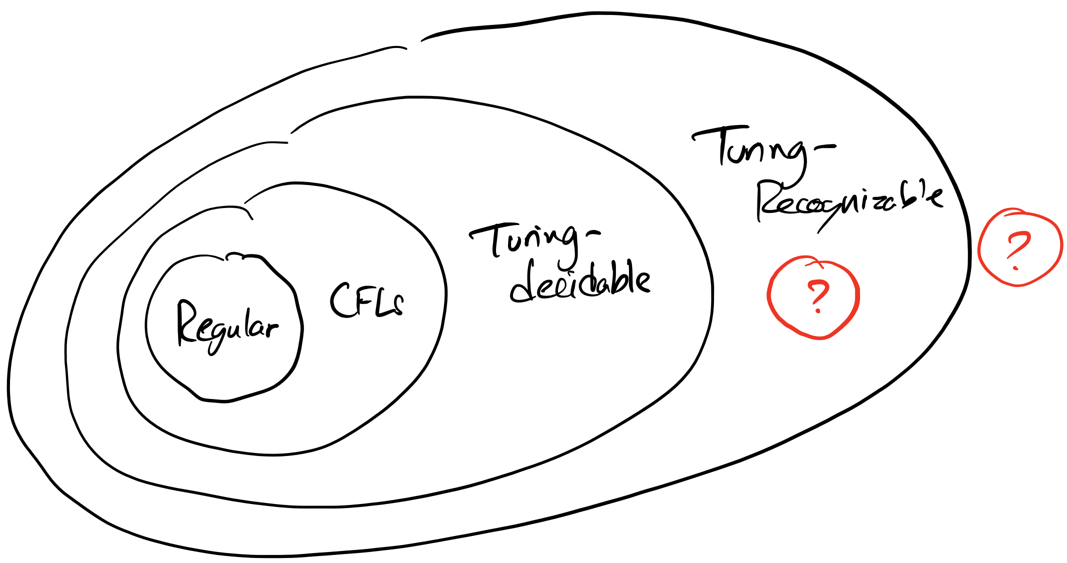
M = "On input w, // Goal: accept if and only if $w \in L(G)$.

1. Write down $\langle G, w\rangle$. implicit in the internal states, "hard-coded"

2. Simulate S on $\langle G, w\rangle$

and accept if and only if S accepts.
(reject otherwise)

Corollary of this machine: Every context-free language is decidable.

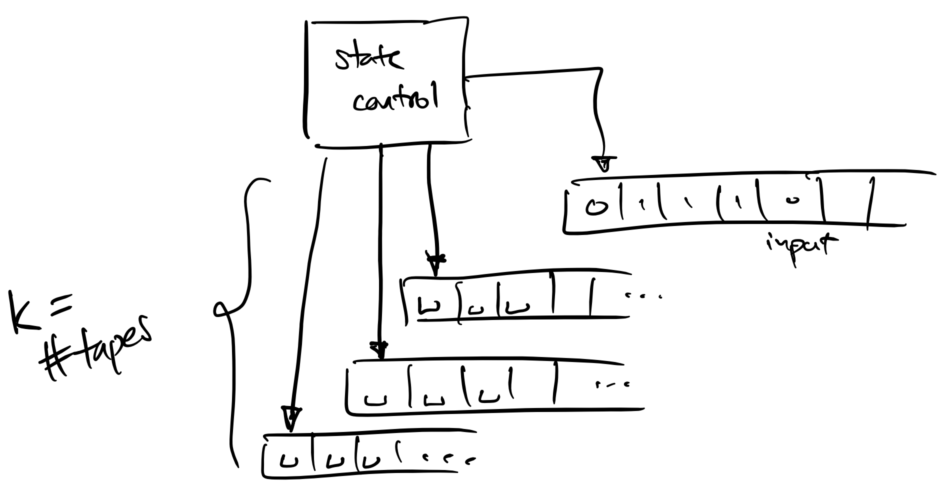


Break -

Back at 2:38

2) TM Variants

2.1) Multitape TM.



Formal def'n of a multitape TM

- exactly the same as an ordinary TM except transition function

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

$\mathbb{Z}^2 = \mathbb{Z} \times \mathbb{Z}$.

$\Gamma^k = \underbrace{\Gamma \times \Gamma \times \Gamma \dots}_{k \text{ times}}$

Theorem: every multi-tape TM has an equivalent 1-tape TM.

Idea: Take an arbitrary multitape TM, show off a regular TM that does the same thing.

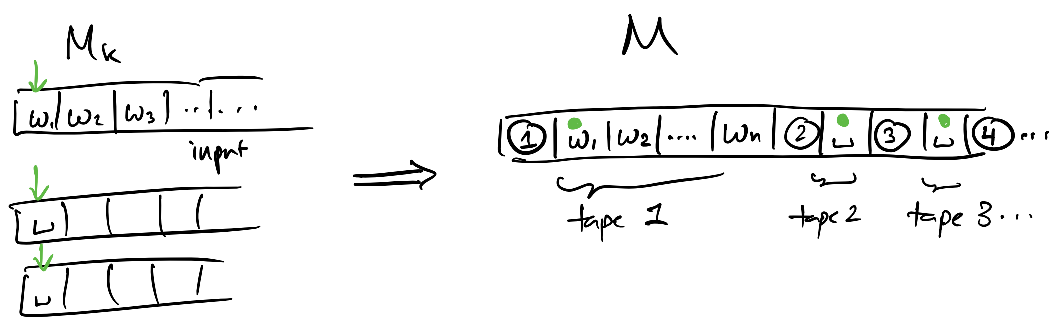
Proof sketch:

Consider the behavior of a multitape TM M_k on input w.

Our equivalent TM works as follows:

M = "on input w:

1. write down the input contents of all tapes in M_k, separated by markers.



2. Mark tape heads with a dot.
3. Simulate M_k on w one step at a time.
4. Whenever we run out of space on a simulated tape:

- pause simulation of M_k
- run subroutine: shift tape contents over one square to make room
- unpause.

5. Accept/reject if our M_k simulation accepts/rejects.

General strategy: show a variant TM recognizes the same languages as a regular TM:

1) Show any instance of the variant has an equivalent TM.

   has the same output on all inputs

2) Show any TM has an equivalent variant TM.

2.2) Nondeterministic TM

Defined exactly as TMs, but with transition function

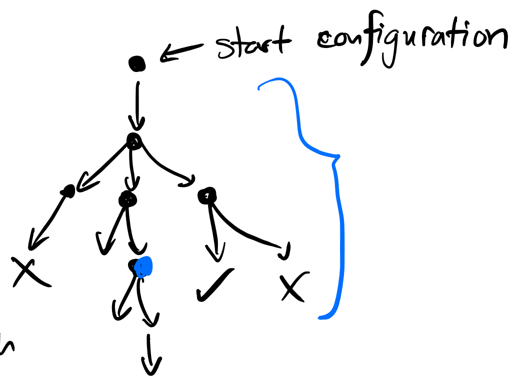
$$\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

different branches move, write, go to different states.

Accept if any branch reaches the accept state.

Theorem: Every NTM has an equivalent TM.

nondeterministic computation.



Goal: make our TM search this tree for an accepting branch.

Proof sketch: Given a NTM T_N, consider the deterministic TM T_D that works as follows.

↑
multitape!

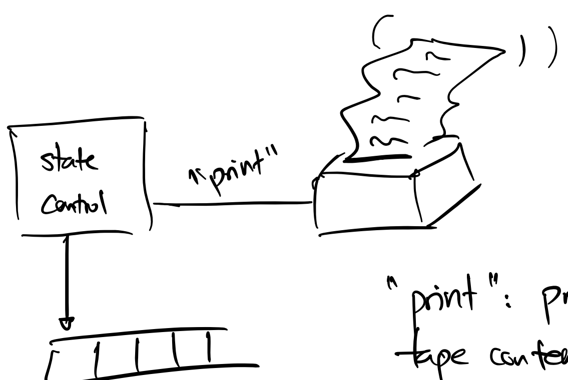
T_D has three tapes:

- input tape stores the input and doesn't change.
- address tape: stores a picture of the tree of computation we've traversed and how to reach each config.
- simulation tape is where we simulate T_N.

T_D does the following on w:

1. Use BFS to explore the tree of computational paths.
2. To get to a new node, use the info in the address tape to simulate T_N to get there.
3. Accept if we ever reach an accepting configuration, reject if we finish the tree and all branches reject.

2.3) Enumerators.



Enumerator:
Same defn as a TM
(w/ magic print command)

"print": print out the tape contents as one line from our printer.

Language enumerated: the set of all strings our enumerator prints.
(can be infinite!

Theorem: A language is Turing-recognizable if and only if some enumerator enumerates it (on ε input).

Proof:

1. Suppose some enumerator **E** enumerates L on input ε.
Consider the TM M that works as follows:

M = "On input w:
1. Ignore the input.
2. Simulate E on ε (E is hardcoded.)
3. Accept if the input w is ever output by the simulation of E."

2. Suppose a TM M recognizes L.
Consider the following enumerator:
Let $s_0, s_1, s_2, \ldots$ be an infinite list of all strings over Σ.

E = "On input ε:
For $i = 0, 1, 2, \ldots$
Simulate M on $s_0, s_1 \ldots s_i$ for i steps.
Print any string that M accepts."

Break: back at 3:40

3. Undecidable & Unrecognizable Languages.

Liar's: "This statement is false." T? ✗ F? ✗

Russell's:

"The barber shaves everyone who doesn't shave themself."

"Let S be the set of all sets that don't contain themselves. Is $S \in S$?"

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

Goal: Assume for contradiction A_{TM} is decidable $\Longrightarrow$ paradox.

Proof: Assume H is a TM that decides A_{TM}.

- $\underline{H(\langle M, w\rangle)}$ accepts/rejects if $\underline{M(w)}$ accepts/doesn't accept.

  "run H on input $\langle M, w\rangle$" "Run M on (w)".

- Very special input:

(*1) $H(\langle M, \langle M\rangle\rangle)$ accept if and only if $M(\langle M\rangle)$ accepts. (reject otherwise.)

```
function foo(string s) {
    return len(s);
}
```

foo("function foo(string s) { return len(s); }")

Define a new machine P.

P = "On input $\langle M\rangle$, simulate $H(\langle M, \langle M\rangle\rangle)$ and return the opposite of $H(\langle M, \langle M\rangle\rangle)$." (*2)

What happens when we run $P(\langle P\rangle)$?

- P will simulate $H(\langle P, \langle P\rangle\rangle)$.
  - If $\underline{P(\langle P\rangle) \text{ accepts}}$, $\overset{(*1)}{}$ $H(\langle P, \langle P\rangle\rangle)$ accepts, $\overset{(*2)}{}$ $\underline{P(\langle P\rangle) \text{ rejects}}$.
  - If $\underline{P(\langle P\rangle) \text{ rejects}}$, $H(\langle P, \langle P\rangle\rangle)$ rejects, $\underline{P(\langle P\rangle) \text{ accepts}}$.

Paradox! Assumption that H decides A_{TM} is false. ∎