Lecture 8

Thursday, 18 February, 2021 09:55

Beyond Regularity: A lightning tour of the Chomsky Hierarchy

Last 3½ weeks: studied languages rep'ed by:
regexes, DFAs, NFAs.
all the same class of languages "regular"
Last time: $\{0^n 1^n \mid n \in \mathbb{N}\}$ is not regular

Q: What other classes of langs are there?
Does one of them include $\{0^n 1^n \mid n \in \mathbb{N}\}$?

×

Context-Free Languages.
rep'd by "context-free grammars" (CFGs)

CFG has variables w/ prod rules.
– while ∃ variable, nondeterministically replace it w/ one of its prod rules

CFG for $\{0^n 1^n \mid n \in \mathbb{N}\}$

$S \to 0S1 \mid \varepsilon$

0011

$S \to 0S1 \to 00S11 \to 00\varepsilon 11 = 0011$

CFG for $\{0^n 1^n\} \cup \{1^n 0^n\}$

$S \to A \mid B$
$A \to 0A1 \mid \varepsilon$
$B \to 1B0 \mid \varepsilon$

1100

$S \to B \to 1B0 \to 11B00 \to 11\varepsilon 00 = 1100$

– machine model "pushdown automaton" (PDA)

NFA w/ a stack (informally)

reading in 0011 pushes 0s to the stack

reading in 0011 pushes 0s to the stack
pops them off while reading 1s
accepts iff stack is empty.

<u>Thm</u>
a language can be rep'd by a CFG iff rep'd by a PDA

Q: Do there exist languages that are <u>not</u> context-free?

Yes. CFLs are closed under union, concat, Kleene*
<u>not</u> closed under intersection, complement.

Ogden's (pumping) lemma: tool for explicitly proving a language is not CF.

$\{0^n 1^n 0^n \mid n \in \mathbb{N}\}$ is <u>not</u> context-free.

——— x ———

Context-Sensitive Language

rep'd by context-sensitive grammars. (CSG)

machine model linear-bounded automata (LBA)

$\{0^n 1^n 0^n\}$ is a CSL

(skip over these)

Q: are there languages not CS?
<u>Yes</u> ...

——— x ———

Chomsky's Hierarchy:

class	machine	grammar
	DFA, NFA	

class	machine	grammar	
regular	DFAs, NFAs, etc.	reg ex	"Type 3"
context-free	PDA	CFG	"Type 2"
context-sensitive	LBA	CSG	"Type 1"
recursively enumerable (Turing recognizable)	TM	unconstrained	"Type 0"

deterministic CFGs >

recursive/decidable >

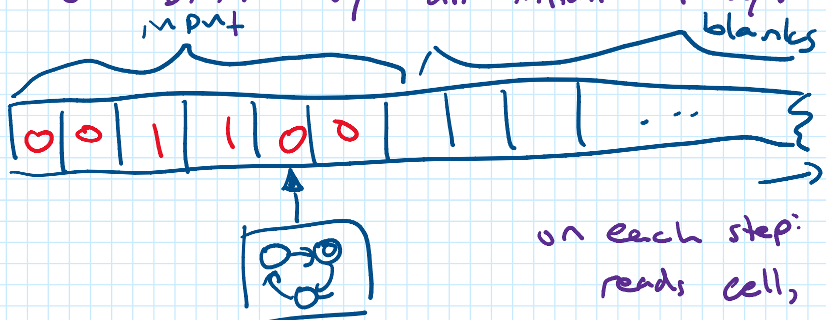
regular $\subseteq$ context-free $\subseteq$ context-sensitive $\subseteq$ rec. en.

"colorless ideas sleep furiously" grammatically correct but meaningless

Turing Machines

written down by Turing in 1936
to capture idea of "general computation"

Informally Turing machine (TM) is
a DFA w/ an infinite read/write tape.



→ ∞ to the right

on each step:
reads cell, writes to cell
changes state, move ← or →.

Q set of states

Q set of states

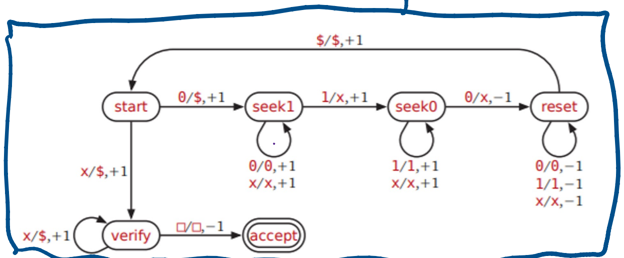
$\square$ special "blank char"

Γ tape alphabet $(\Gamma \supseteq \Sigma \cup \{\square\})$

start, accept, reject $\in Q$

ex $L_{0^n1^n0^n} = \{0^n1^n0^n \mid n \in \mathbb{N}\}$

TM w/ $\Gamma = \{0, 1, \square, \$, x\}$



δ(p , a) = (q , b, Δ)	explanation
δ(start , 0) = (seek1, $, +1)	mark first 0 and scan right
δ(start , x) = (verify, $, +1)	looks like we're done, but let's make sure
δ(seek1, 0) = (seek1, 0, +1)	scan rightward for 1
δ(seek1, x) = (seek1, x, +1)	
δ(seek1, 1) = (seek0, x, +1)	mark 1 and continue right
δ(seek0, 1) = (seek0, 1, +1)	scan rightward for 0
δ(seek0, x) = (seek0, x, +1)	
δ(seek0, 0) = (reset , x, +1)	mark 0 and scan left
δ(reset , 0) = (reset , 0, −1)	scan leftward for $
δ(reset , 1) = (reset , 1, −1)	
δ(reset , x) = (reset , x, −1)	
δ(reset , $) = (start , $, +1)	step right and start over
δ(verify, x) = (verify , $, +1)	scan right for any unmarked symbol
δ(verify, □) = (accept, □, −1)	success!

current state

current tape

example computations

(start, 001100)
⇒ (seek1, $01100)
⇒ (seek1, $01100)
⇒ (seek0, $0x100)
⇒ (seek0, $0x100)
⇒ (reset, $0x1x0)
⇒ (reset, $0x1x0)
⇒ (reset, $0x1x0)
⇒ (reset, $0x1x0)
⇒ (start, $0x1x0)
⇒ (seek1, $$x1x0)
⇒ (seek1, $$x1x0)
⇒ (seek0, $$xxx0)
⇒ (seek0, $$xxx0)
⇒ (reset, $$xxxx)
⇒ (reset, $$xxxx)
⇒ (reset, $$xxxx)
⇒ (reset, $$xxxx)
⇒ (start, $$xxxx)
⇒ (verify, $$$xxx)
⇒ (verify, $$$$xx)
⇒ (verify, $$$$$x)
⇒ (verify, $$$$$$□)
⇒ (accept, $$$$$$) ⇒ **accept!**

(start, 00100)
⇒ (seek1, $0100)
⇒ (seek1, $0100)
⇒ (seek0, $0x00)
⇒ (reset, $0xx0)
⇒ (reset, $0xx0)
⇒ (reset, $0xx0)
⇒ (start, $0xx0)
⇒ (seek1, $$xx0)
⇒ (seek1, $$xx0)
⇒ (seek1, $$xx0) ⇒ **reject!**

start $\xrightarrow{0/\$,+1}$ seek1

on reading 0, write \$, move →, switch to state seek1

Interpretation? <u>Pseudocode</u>

IsStringIn $L_{0^n1^n0^n}$(w):
    i ← 0
    while w[i] = 0 :
        // mark 010 subseq.

```
while w[i] = 0 :
    // mark 010 subseq.
    w[i] ← $
    while w[i] = 0 or w[i] = x
        i ← i+1
    reject if w[i] ≠ 1
    w[i] ← x
    while w[i] = 1 or w[i] = x
        i ← i+1
    reject if w[i] ≠ 0
     w[i] ← x

    // reset
    while w[i] ≠ $
        i ← i-1
    i ← i+1
// verify all symbols marked
 while w[i] = x
 accept iff w[i] = □
```

IsStringInL0n1n0n(w):

    while there exist unmarked 0s :
        try to mark first 010 subseq
        reject if fail.
    accept iff all symbols are marked.

---

why is this OK?

Can simulate a TM using:

- Minecraft

- Powerpoint
- Baba Is You
- more

all of these coded up in

C++, Java, Python, Lua, etc...

→ simulated on hardware via

assembly + RAM

actually, every assembly + RAM machine can be simulated via TM.

Church-Turing Thesis:

<u>Every</u> general computer is equivalent to a TM (in terms of "what is computable).

not just decision problems!

can build a TM for

w + x → output the sum!

Two Big Qs:

1) Is C-T a theorem?

<u>No</u>. Not mathematically proven

(part of why we don't define precisely "general computer")

[turns out many previously proposed GC's] computer"]

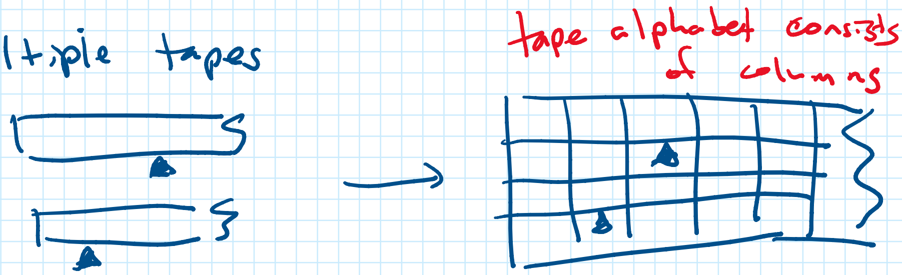
[turns out many previously proposed GC's are "polynomially equivalent" to TMs
(intuitively, TMs also capture "efficiency of comp")
Q.C.s broke this.]

2) Why TMs specifically?

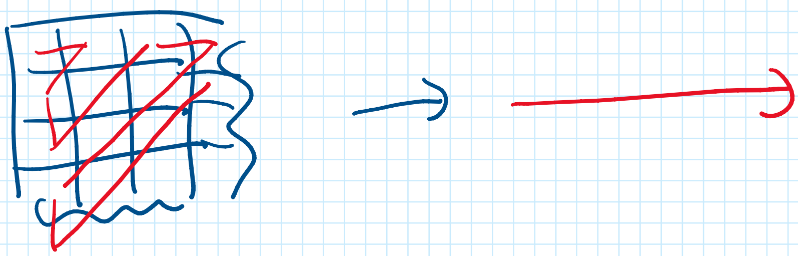
not really in practice, but useful (kinda) when proving mathematical statements

→ which TM?

– multiple tapes



– 2d tape



every TM variant is eq to TMs

→ pick the variant best for proof technique.

(idea used previously, e.g. reg. lang. complement much easier on DFAs than NFAs/regex)

———— X ————

———— X ————

Q: are there languages that are <u>not</u> r.e.?

Yes. One way to see this is purely counting.

\# binary langs? $\left|P(\{0,1\}^*)\right|$ = uncountably infinite.

$$
\begin{aligned}
\#\ \text{TMs?} &= \#\ \text{Python programs} \\
&= \#\ \text{C programs} \\
&= \text{etc.} \\
&= \left|\{0,1\}^*\right| \neq \left|P(\{0,1\}^*)\right|
\end{aligned}
$$

every program is storable as source code in binary on your computer.

$\longrightarrow$ $\exists$ problems not solvable by "general computer"