Beyond Regularity: A lightning tour of the Chomsky Hierarchy

Last 3½ weeks: studied languages rep'd by:

- regular languages
- DFA
- NFA

all the same class of languages  "regular"

Last time: \( \{0^n1^n \mid n \in \mathbb{N}\} \) is not regular

Q: What other classes of langs are there?

Does one of them include \( \{0^n1^n \mid n \in \mathbb{N}\} \)?

\[ \times \]

Context-Free Languages.

rep'd by "context-free grammars" (CFGs)

- CFG has variables w/ prod rules

- while \( \exists \) variable, nondeterministically replace it w/ one of its prod rules

CFG for \( \{0^n1^n \mid n \in \mathbb{N}\} \):

\[
S \rightarrow 0S1 | \varepsilon
\]

CFG for \( \{0^n1^n \mid n \in \mathbb{N}\} \):

\[
S \rightarrow A | B
A \rightarrow 0A | \varepsilon
B \rightarrow 1B | \varepsilon
\]

- machine model "pushdown automaton" (PDA)

NFA w/ a stack (informally)

reading in \( 0011 \) pushes 0s to the stack

\[ \Rightarrow 003 \]

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

Then a language can be rep'd by a CFG iff rep'd by a PDA.

Q: Do there exist languages that are not context-free?
Yes. CFLs are closed under union, concat, Kleene*,
not closed under intersection, complement.

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

\[ \exists 0^n 1^n 0^n \mid n \in \mathbb{N} \quad \Rightarrow \quad \text{not context-free.} \]

Context-Sensitive Language
rep'd by context-sensitive grammars (CSG)
machine model linear-bounded automata (LBA)

\[ \exists 0^n 1^n 0^n \mid n \in \mathbb{N} \quad \Rightarrow \quad \text{CSL} \]

(skip over these)

Q: are there languages not CS?
Yes...

Chomsky's Hierarchy:

class | machine | grammar

[Diagram with classes and machines labeled]
<table>
<thead>
<tr>
<th>Class</th>
<th>Machine</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>deterministic</td>
<td>DFAs, NFAs, etc.</td>
<td>regular &quot;Type 3&quot;</td>
</tr>
<tr>
<td>CFs</td>
<td>PDA</td>
<td>CFG</td>
</tr>
<tr>
<td>context-free</td>
<td>LBA</td>
<td>&quot;Type 2&quot;</td>
</tr>
<tr>
<td>context-sensitive</td>
<td>TM</td>
<td>CSG</td>
</tr>
<tr>
<td>recursively</td>
<td>unconstrained</td>
<td>&quot;Type 1&quot;</td>
</tr>
<tr>
<td>enumerable</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turing recogniz.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

regular $\subseteq$ context-free $\subseteq$ context-sensitive $\subseteq$ recursive

"colorless ideas sleep furiously" grammatically correct but meaningless

---

Turing Machines

- Written down by Turing in 1936
- To capture idea of "general computation"

Informally, a Turing machine (TM) is:

- A DFA w/ an infinite read/write tape.

- A set of states $Q$.

- On each step:
  - Reads cell, writes to cell
  - Changes state, moves L or R.

- Blanks:

- Input:

- Output:

---
Q: set of states

\[ (*) \text{ special "blank char"} \]

\( \Gamma \): tape alphabet \( (\Gamma = \Sigma \cup \{ \delta \}) \)

Start, accept, reject \( \in Q \)

Example computations

\[ L_{010^n} = \{ 0^n1^0^n \mid n \in \mathbb{N}^3 \} \]

TM \( w_1 \Gamma = \{ 0, 1, \delta, $, x \} \)

\( \delta \): current state | current tape

\( \delta( p, a) = (q, b, \Delta) \): explanation

\( \delta(\text{start}, 0) = (\text{seek1}, $, +1) \)

- mark first \( \delta \) and scan right

\( \delta(\text{start}, x) = (\text{verify}, $, +1) \)

- looks like we’re done, but let’s make sure

\( \delta(\text{seek1}, 0) = (\text{seek1}, 0, +1) \)

- scan rightward for 1

\( \delta(\text{seek1}, x) = (\text{seek1}, x, +1) \)

- mark 1 and continue right

\( \delta(\text{seek0}, 1) = (\text{seek0}, 1, +1) \)

- scan rightward for $ 

\( \delta(\text{seek0}, x) = (\text{seek0}, x, +1) \)

- mark $ and scan left

\( \delta(\text{reset}, 0) = (\text{reset}, 0, -1) \)

- scan leftward for $ 

\( \delta(\text{reset}, 1) = (\text{reset}, 1, -1) \)

- 

\( \delta(\text{reset}, x) = (\text{reset}, x, -1) \)

- 

\( \delta(\text{reset}, $) = (\text{start}, $, +1) \)

- step right and start over

\( \delta(\text{verify}, 0) = (\text{verify}, 0, +1) \)

- scan right for any unmarked symbol

\( \delta(\text{verify}, \delta) = (\text{accept}, \delta, -1) \)

- success!

\( \delta(\text{accept}, $) = (\text{accept}, $) \)

- accept!

Interpretation?

Pseudocode

Is \( \text{StringIn} \ L_{010^n}(w) \):

\( i \in \emptyset \)

while \( w[i] = 0 \):

// mark \( 00 \) subseq.
while \( w[i] = 0 \):
  // mark old subseq.
  \( w[i] \leftarrow \$ \)
  while \( w[i] = 0 \) or \( w[i] = x \)
    \( i \leftarrow i + 1 \)
  reject if \( w[i] = 1 \)
  \( w[i] \leftarrow x \)
  while \( w[i] = 1 \) or \( w[i] = x \)
    \( i \leftarrow i + 1 \)
  reject if \( w[i] = 0 \)
  \( w[i] \leftarrow x \)

// reset
while \( w[i] \neq \$ \)
  \( i \leftarrow i - 1 \)
  \( i \leftarrow i + 1 \)

// verify all symbols marked
while \( w[i] = x \)
  accept iff \( w[i] = \square \)

---

IsStrongInLondon\( (w) \):
  while there exist unmarked Os:
    try to mark first old subseq
    reject if fail.
  accept iff all symbols are marked.

---

why is this OK?

Can simulate a TM using:
  - Minecraft
- PowerPoint

- Bah, Is You

- All of these coalesced 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:

Every general computer is equivalent to

a TM (in terms of “what is computable”).

Not just decision problems!

Can build a TM for

\[ w + x \rightarrow \text{output the sum!} \]

Two Big Qs:

1) Is CT a theorem?
   
   No. Not mathematically proven
   
   (part of why we don’t define precisely “general computer”)

   Turns out, many previously proposed GCs
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/parsers)
Q: are there languages that are not r.e.?
Yes. One way to see this is purely counting.

# binary langs?
\[ |P(\{0,1\}^*)| = \text{uncountably infinite.} \]

# TMs?

= # Python programs
  = # C programs
  = etc.

= \[ |\{0,1\}^*| \neq |P(\{0,1\}^*)| \]

\[ \uparrow \]
every program is storable as source code in binary on your computer.

\[ \rightarrow \] \exists problems not solvable by "general computer"