

# Foundations of Computer Science

## Lecture 25

### Context Free Grammars (CFGs)

Solving a Problem by “Listing Out” the Language

Rules for CFGs

Parse Trees

Pushdown Automata



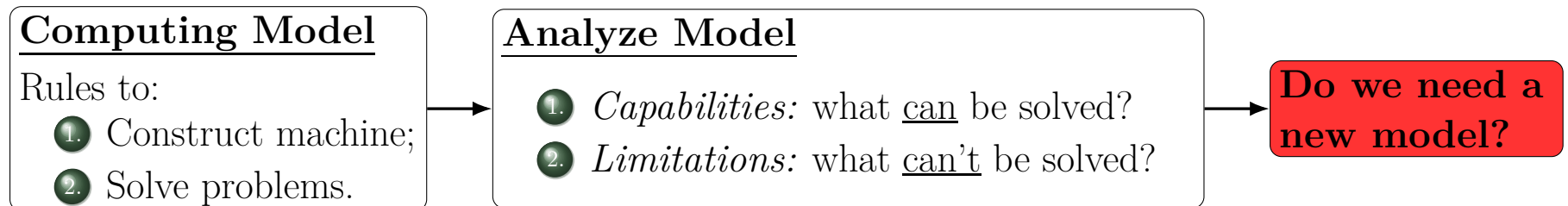
# Last Time

---

DFAs: State transitioning machines which can be implemented using basic technology.

Powerful: can solve any regular expression.

(Finite sets, complement, union, intersection, concatenation, Kleene-star).

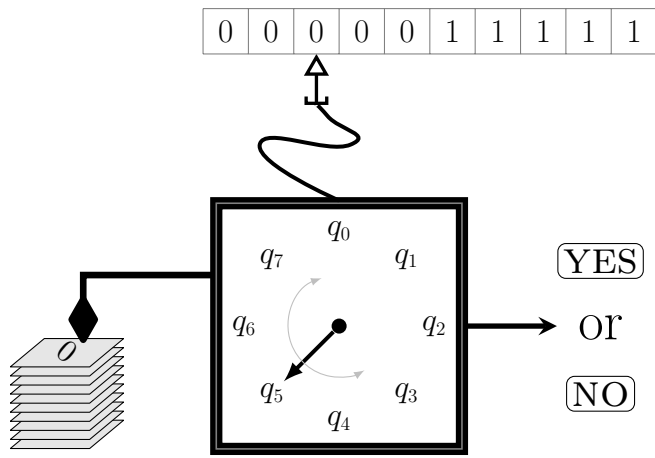


DFAs fail at so simple a problem as equality.

- That's not acceptable.
- We need a more powerful machine.

# Adding Memory

DFAs have no scratch paper. It's hard to compute entirely in your head.



**Stack Memory.** Think of a file-clerk with a stack of papers.  
The clerk's capabilities:

- see the top sheet;
- remove the top sheet (*pop*)
- *push* something new onto the top of the stack.
- no access to inner sheets without removing top.

DFA with a stack is a *pushdown automaton (PDA)*

How does the stack help to solve  $\{0^n 1^n \mid n \geq 0\}$ ?

- 1: When you read in each 0, write it to the stack.
- 2: For each 1, pop the stack. At the end if the stack is empty, ACCEPT.

The memory allows the automaton to “remember”  $n$ .

# Today: Context Free Grammars (CFGs)

---

- 1 Solving a problem by listing out the language.
- 2 Rules for Context Free Grammars (CFG).
- 3 Examples of Context Free Grammars.
  - English.
  - Programming.
- 4 Proving a CFG solves a problem.
- 5 Parse Trees.
- 6 Pushdown Automata and non context free languages.

# Recursively Defined Language: Listing a Language.

---

$$\mathcal{L}_{0^n 1^n} = \{0^n 1^n \mid n \geq 0\}$$

- ①  $\varepsilon \in \mathcal{L}_{0^n 1^n}$ . [basis]
- ②  $x \in \mathcal{L}_{0^n 1^n} \rightarrow 0 \bullet x \bullet 1 \in \mathcal{L}_{0^n 1^n}$ . [constructor rule]
- ③ Nothing else is in  $\mathcal{L}_{0^n 1^n}$ . [minimality]

To test if  $0010 \in \mathcal{L}_{0^n 1^n}$ : generate strings in order of length and test each for a match:

$$\varepsilon \rightarrow 01 \rightarrow 0011 \rightarrow 000111$$

NO

A Context Free Grammar is like a recursive definition.

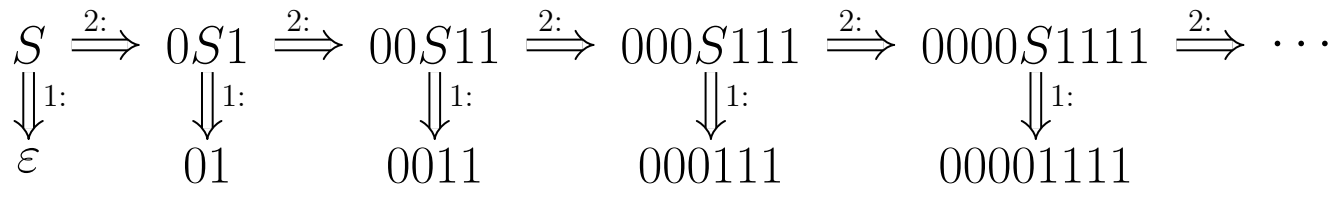
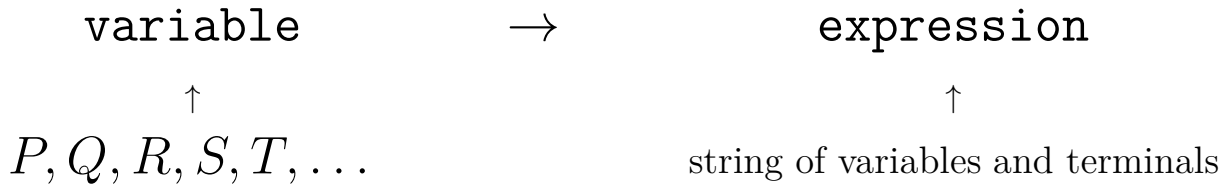
$$\begin{array}{l} 1: S \rightarrow \varepsilon \\ 2: S \rightarrow 0S1 \end{array} \quad \left( \begin{array}{l} \varepsilon \in \mathcal{L}_{0^n 1^n} \\ x \in \mathcal{L}_{0^n 1^n} \rightarrow 0 \bullet x \bullet 1 \in \mathcal{L}_{0^n 1^n} \end{array} \right)$$

# Rules for Context Free Grammars (CFGs)

Production rules of the CFG:

- 1:  $S \rightarrow \epsilon$
- 2:  $S \rightarrow 0S1$

Each production rule has the form



- 1: Write down the start variable (form the first production rule, typically  $S$ ).
- 2: Replace *one* variable in the current string with the expression from a production rule that *starts* with that variable on the left.
- 3: Repeat step 2 until no variables remain in the string.

“Replace **variable** with **expression**, no matter where (independent of context)”

Shorthand:  $1: S \rightarrow \epsilon \mid 0S1$

# Language of Equality, $CFG_{bal}$

$$CFG_{bal} \quad 1: S \rightarrow \varepsilon \mid 0S1S \mid 1S0S$$

A *derivation* of 0110 in  $CFG_{bal}$  (each step is called an inference).

$$S \xRightarrow{1} 0S1S \xRightarrow{1} 0S11S0S \xRightarrow{1} 0\varepsilon11S0S \xRightarrow{*} 0110$$

Notation:

$$S \xRightarrow{*} 0110 \quad (\xRightarrow{*} \text{ means "A derivation starting from } S \text{ yields } 0110")$$

Distinguish  $S$  from a *mathematical* variable (e.g.  $x$ ),

$$0S1S \quad \text{versus} \quad 0x1x$$

Two  $S$ 's are replaced independently. Two  $x$ 's must be the same, e.g.  $x = 11$ .

**Pop Quiz.** Determine if each string can be generated and if so, give a derivation.

- (a) 0011
- (b) 0110
- (c) 00011
- (d) 010101

- 1:  $S \rightarrow \varepsilon \mid T_0T_1 \mid T_0A$
- 2:  $X \rightarrow T_0T_1 \mid T_0A$
- 3:  $A \rightarrow XT_1$
- 4:  $T_0 \rightarrow 0$
- 5:  $T_1 \rightarrow 1$

Give an informal description for the CFL of this CFG.

# A CFG for English

---

- 1:  $S \rightarrow \langle \text{phrase} \rangle \langle \text{verb} \rangle$
- 2:  $\langle \text{phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle$
- 3:  $\langle \text{article} \rangle \rightarrow A_{\square} \mid \text{The}_{\square}$
- 4:  $\langle \text{noun} \rangle \rightarrow \text{cat}_{\square} \mid \text{dog}_{\square}$
- 5:  $\langle \text{verb} \rangle \rightarrow \text{walks.} \mid \text{runs.} \mid \text{walks.}_{\square} S \mid \text{runs.}_{\square} S$

$$\begin{aligned} S &\xrightarrow{1:} \langle \text{phrase} \rangle \langle \text{verb} \rangle \\ &\xrightarrow{5:} \langle \text{phrase} \rangle \text{walks.} \\ &\xrightarrow{2:} \langle \text{article} \rangle \langle \text{noun} \rangle \text{walks.} \\ &\xrightarrow{3:} A_{\square} \langle \text{noun} \rangle \text{walks.} \\ &\xrightarrow{4:} A_{\square} \text{cat}_{\square} \text{walks.} \end{aligned}$$

**Pop Quiz.** Give a derivation for:  $A_{\square} \text{cat}_{\square} \text{runs.}_{\square} \text{The}_{\square} \text{dog}_{\square} \text{walks.}$



# A CFG for Programming

---

- 1:  $S \rightarrow \langle \text{stmt} \rangle ; S \mid \langle \text{stmt} \rangle ;$
- 2:  $\langle \text{stmt} \rangle \rightarrow \langle \text{assign} \rangle \mid \langle \text{declare} \rangle$
- 3:  $\langle \text{declare} \rangle \rightarrow \text{int} \_ \langle \text{variable} \rangle$
- 4:  $\langle \text{assign} \rangle \rightarrow \langle \text{variable} \rangle = \langle \text{integer} \rangle$
- 5:  $\langle \text{integer} \rangle \rightarrow \langle \text{integer} \rangle \langle \text{digit} \rangle \mid \langle \text{digit} \rangle$
- 6:  $\langle \text{digit} \rangle \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
- 7:  $\langle \text{variable} \rangle \rightarrow x \mid x \langle \text{variable} \rangle$

**Pop Quiz.** Give derivations for these snippets of code.

- (a) `int _ x ; int _ xx ; x = 22 ; xx = 8 ;`
- (b) `x = 8 ; int _ x ;`
- (c) `int _ x ; xx = 8 ;`

# Constructing a CFG to Solve a Problem

---

$\mathcal{L}_{\text{bal}} = \{\text{strings with an equal number of 1's and 0's}\}.$

$$\begin{aligned} 001011010110 &= 0 \bullet \mathbf{0101} \bullet 1 \bullet \mathbf{010110} \\ &\quad \uparrow \qquad \qquad \uparrow \\ &\quad \text{in } \mathcal{L}_{\text{bal}} \qquad \text{in } \mathcal{L}_{\text{bal}} \\ &= 0S1S \qquad \qquad \qquad (S \text{ represents "a string in } \mathcal{L}_{\text{bal}}\text{"}) \end{aligned}$$

Every large string in  $\mathcal{L}_{\text{bal}}$  can be obtained (recursively) from smaller ones.

$$S \rightarrow \varepsilon \mid 0S1S \mid 1S0S.$$

We must *prove* that:

- ❶ Every string generated by this CFG is in  $\mathcal{L}_{\text{bal}}$ ?
- ❷ Every string in  $\mathcal{L}_{\text{bal}}$  can be derived by this grammar?

# Proving a CFG Solves a Problem

---

$$\mathcal{L}_{\text{bal}} = \{\text{strings with an equal number of 1's and 0's}\}$$

$$S \rightarrow \varepsilon \mid 0S1S \mid 1S0S$$

(i) *Every derivation in  $\text{CFG}_{\text{bal}}$  generates a string in  $\mathcal{L}_{\text{bal}}$ .*

Strong induction on the length of the derivation (number of production rules invoked).

Base Case. length-1 derivation gives  $\varepsilon$ .

Induction. The derivation starts in one of two ways:

$$S \rightarrow 0 \underset{\substack{\downarrow^* \\ x}}{S} 1 \underset{\substack{\downarrow^* \\ y}}{S} \rightarrow \dots \quad \text{or} \quad S \rightarrow 1 \underset{\substack{\downarrow^* \\ x}}{S} 0 \underset{\substack{\downarrow^* \\ y}}{S} \rightarrow \dots$$

The derivations of  $x$  and  $y$  are shorter.

By the induction hypothesis,  $x, y \in \mathcal{L}_{\text{bal}}$ , so the final strings are in  $\mathcal{L}_{\text{bal}}$ .

(ii) *Every string in  $\mathcal{L}_{\text{bal}}$  can be derived within  $\text{CFG}_{\text{bal}}$ .*

Strong induction on the length of the string.

Base case: length-1 string,  $\varepsilon$ .

Induction. *Any* string  $w$  in  $\mathcal{L}_{\text{bal}}$  has one of two forms:

$$w = 0w_11w_2 \quad \text{or} \quad w = 1w_10w_2,$$

where  $w_1, w_2 \in \mathcal{L}_{\text{bal}}$  and have smaller length.

By the induction hypothesis,  $S \xRightarrow{*} w_1$  and  $S \xRightarrow{*} w_2$ , so  $S \xRightarrow{*} w$ .

**Practice.** Exercise 25.5.

# Union, Concatenation, Kleene-star

---

$$\mathcal{L}_1 = \{0^n 1^n \mid n \geq 0\}$$

$$A \rightarrow \varepsilon \mid 0A1$$

$$\mathcal{L}_2 = \{1^n 0^n \mid n \geq 0\}$$

$$B \rightarrow \varepsilon \mid 1B0$$

$$\begin{aligned} \mathcal{L}_1 \cup \mathcal{L}_2 : \quad & 1: S \rightarrow A \mid B \\ & 2: A \rightarrow \varepsilon \mid 0A1 \\ & 3: B \rightarrow \varepsilon \mid 1B0 \end{aligned}$$

$$\begin{aligned} \mathcal{L}_1 \bullet \mathcal{L}_2 : \quad & 1: S \rightarrow AB \\ & 2: A \rightarrow \varepsilon \mid 0A1 \\ & 3: B \rightarrow \varepsilon \mid 1B0 \end{aligned}$$

**Kleene-star.**  $\mathcal{L}_1^*$  is generated by the CFG

$$\begin{aligned} 1: S &\rightarrow \varepsilon \mid SA \\ 2: A &\rightarrow \varepsilon \mid 0A1 \end{aligned}$$

← generates  $A^i$

← each  $A$  becomes a  $0^n 1^n$

**Example 25.2.** CFGs can implement DFAs, and so are strictly more powerful.

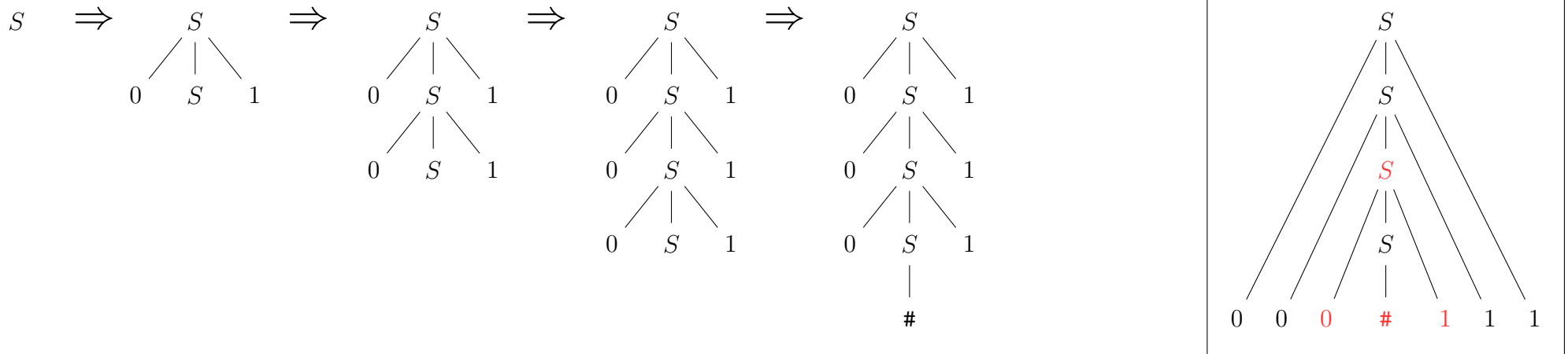
# Parse Trees

$$S \rightarrow \# \mid 0S1$$

Here is a derivation of 000#111,

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000\#111$$

The *parse tree* gives more information than a derivation



Clearly shows how a substring belongs to the language of its parent variable.

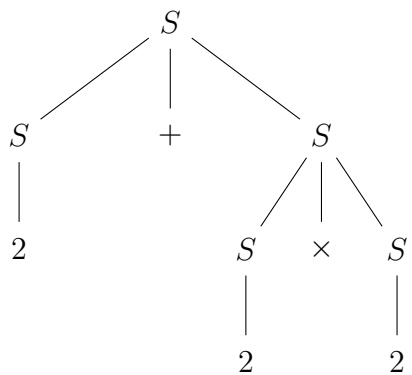
# CFG for Arithmetic

$$S \rightarrow S + S \mid S \times S \mid (S) \mid 2$$

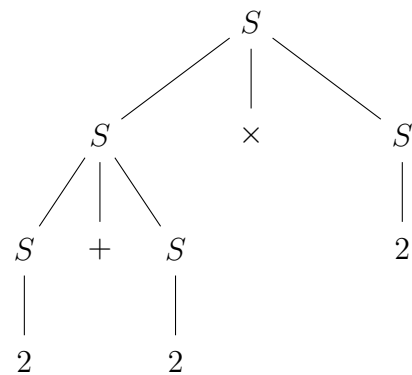
(the terminals are +, ×, (, ) and 2)

Two derivations of 2 + 2 × 2 along with parse trees,

$$S \Rightarrow S + S \Rightarrow S + S \times S \xRightarrow{*} 2 + 2 \times 2 \qquad S \Rightarrow S \times S \Rightarrow S + S \times S \xRightarrow{*} 2 + 2 \times 2$$



(multiply 2 × 2 and add to 2)



(add 2 + 2 and multiply by 2)

- Parse tree ↔ How you interpret the string.
- Different parse trees ↔ different meanings.
- **BAD!** We want unambiguous meaning  
programs, html-code, math, English, ...

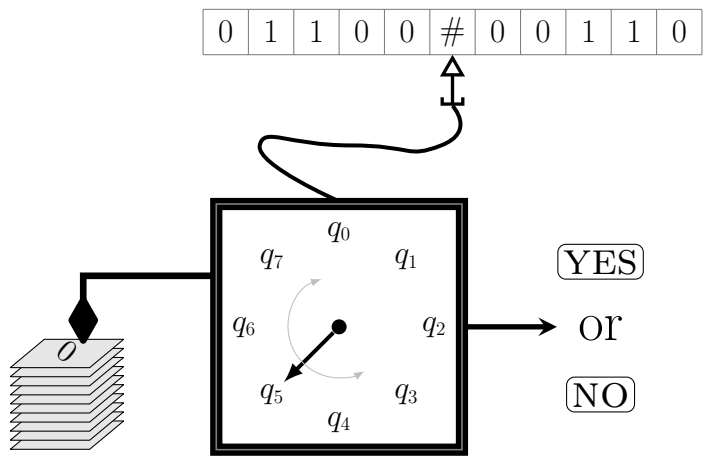
Unambiguous grammar

- 1:  $S \rightarrow P \mid S + P$
- 2:  $P \rightarrow T \mid P \times T$
- 3:  $T \rightarrow 2 \mid (S)$

# Pushdown Automata: DFAs with Stack Memory

$$\mathcal{L} = \{w\#w^R \mid w \in \{0, 1\}^*\}$$

$$S \rightarrow \# \mid 0S0 \mid 1S1$$



DFA with stack memory (push, pop, read).

Push the first half of the string (before #).

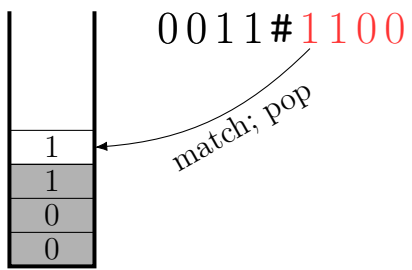
For each bit in the second half, pop the stack and compare.

DFAs with stack memory closely related to CFGs.

# Non Context Free

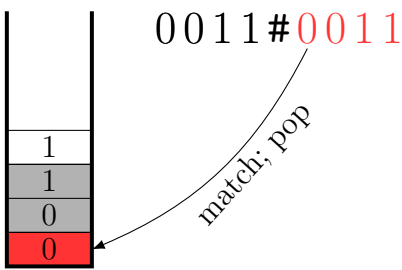
- $\{w#w\}$
  - $\{0^n 1^n 0^n\}$
  - $\{0^{n^2}\}, \{0^n 1^{n^2}\}$
  - $\{0^{2^n}\}, \{0^n 1^{2^n}\}$
- repetition  
multiple-equality  
squaring  
exponentiation

$w#w^R$



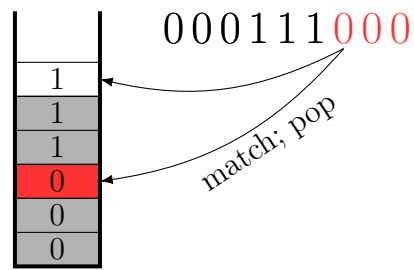
0011 is pushed.  
DFA matches 1100 by popping.

$w#w$



0011 is pushed.  
DFA needs bottom-access to match.

$0^n 1^n 0^n$



000111 is pushed.  
DFA needs random access to match.

The file clerk who only has access to the top of his *stack* of papers has fundamentally less power than the file clerk who has a *filing cabinet* with access to all his papers.

**We need a new model,** one with Random Access Memory (RAM).