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).

Computing Model
Rules to:
1. Construct machine;
2. Solve problems.

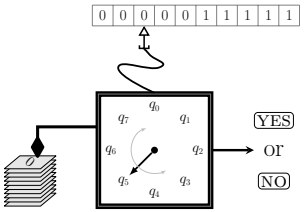
→ **Analyze Model**
1. *Capabilities:* what can be solved?
2. *Limitations:* what can't be solved?

→ **Do we need a new model?**

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.

$$\mathcal{L}_{0^n1^n} = \{0^{\bullet n}1^{\bullet n} \mid n \geq 0\}$$

1. $\varepsilon \in \mathcal{L}_{0^n1^n}$. **[basis]**
2. $x \in \mathcal{L}_{0^n1^n} \to 0 \bullet x \bullet 1 \in \mathcal{L}_{0^n1^n}$. **[constructor rule]**
3. Nothing else is in $\mathcal{L}_{0^n1^n}$. **[minimality]**

To test if $0010 \in \mathcal{L}_{0^n1^n}$: generate strings in order of length and test each for a match:

$$\varepsilon \;\to\; 01 \;\to\; 0011 \;\to\; 000111$$

NO

A Context Free Grammar is like a recursive definition.

$$\begin{array}{ll} 1\text{:}\ S \to \varepsilon & \left(\begin{array}{l}\varepsilon \in \mathcal{L}_{0^n1^n} \\ x \in \mathcal{L}_{0^n1^n} \to 0 \bullet x \bullet 1 \in \mathcal{L}_{0^n1^n}\end{array}\right) \\ 2\text{:}\ S \to 0S1 & \end{array}$$

Rules for Context Free Grammars (CFGs)

Production rules of the CFG:

$$\begin{array}{ll} 1\text{:} & S \to \varepsilon \\ 2\text{:} & S \to 0S1 \end{array}$$

Each production rule has the form

$$\begin{array}{ccc} \texttt{variable} & \to & \texttt{expression} \\ \uparrow & & \uparrow \\ P, Q, R, S, T, \ldots & & \text{string of variables and terminals} \end{array}$$

$$\begin{array}{ccccccccc} S & \overset{2:}{\Longrightarrow} & 0S1 & \overset{2:}{\Longrightarrow} & 00S11 & \overset{2:}{\Longrightarrow} & 000S111 & \overset{2:}{\Longrightarrow} & 0000S1111 \overset{2:}{\Longrightarrow} \cdots \\ \Big\downarrow{\scriptstyle 1:} & & \Big\downarrow{\scriptstyle 1:} & & \Big\downarrow{\scriptstyle 1:} & & \Big\downarrow{\scriptstyle 1:} & & \Big\downarrow{\scriptstyle 1:} \\ \varepsilon & & 01 & & 0011 & & 000111 & & 00001111 \end{array}$$

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: $1\text{:}\ S \to \varepsilon \mid 0S1$

Language of Equality, CFG_{bal}

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

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

$$S \overset{1:}{\Rightarrow} 0S1S \overset{1:}{\Rightarrow} 0S11S0S \overset{1:}{\Rightarrow} 0\varepsilon11S0S \overset{*}{\Rightarrow} 0110$$

Notation:

$S \overset{*}{\Rightarrow} 0110$ ($\overset{*}{\Rightarrow}$ means "A derivation starting from S yields 0110")

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

$0S1S$ versus $0x1x$

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.

1. 0011
2. 0110
3. 00011
4. 010101

Give an informal description for the CFL of this CFG.

$$\begin{array}{ll} 1\text{:} & S \to \varepsilon \mid T_0T_1 \mid T_0A \\ 2\text{:} & X \to T_0T_1 \mid T_0A \\ 3\text{:} & A \to XT_1 \\ 4\text{:} & T_0 \to 0 \\ 5\text{:} & T_1 \to 1 \end{array}$$

A CFG for English

$$\begin{array}{lll} 1\text{:} & S & \to \texttt{<phrase><verb>} \\ 2\text{:} & \texttt{<phrase>} & \to \texttt{<article><noun>} \\ 3\text{:} & \texttt{<article>} & \to \text{A}\sqcup \mid \text{The}\sqcup \\ 4\text{:} & \texttt{<noun>} & \to \text{cat}\sqcup \mid \text{dog}\sqcup \\ 5\text{:} & \texttt{<verb>} & \to \text{walks.} \mid \text{runs.} \mid \text{walks.}\sqcup S \mid \text{runs.}\sqcup S \end{array}$$

$$\begin{array}{ll} S & \overset{1:}{\Rightarrow} \texttt{<phrase><verb>} \\ & \overset{5:}{\Rightarrow} \texttt{<phrase>}\text{walks.} \\ & \overset{2:}{\Rightarrow} \texttt{<article><noun>}\text{walks.} \\ & \overset{3:}{\Rightarrow} \text{A}\sqcup\texttt{<noun>}\text{walks.} \\ & \overset{4:}{\Rightarrow} \text{A}\sqcup\text{cat}\sqcup\text{walks.} \end{array}$$

Pop Quiz. Give a derivation for: $\text{A}\sqcup\text{cat}\sqcup\text{runs.}\sqcup\text{The}\sqcup\text{dog}\sqcup\text{walks.}$

A CFG for Programming

1: $S \rightarrow$ `<stmt>;`S | `<stmt>;`
2: `<stmt>` $\rightarrow$ `<assign>` | `<declare>`
3: `<declare>` $\rightarrow$ `int␣<variable>`
4: `<assign>` $\rightarrow$ `<variable>=<integer>`
5: `<integer>` $\rightarrow$ `<integer><digit>` | `<digit>`
6: `<digit>` $\rightarrow$ `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`
7: `<variable>` $\rightarrow$ `x` | `x<variable>`

Pop Quiz. Give derivations for these snippets of code.

1. `int␣x;int␣xx;x=22;xx=8;`
2. `x=8;int␣x;`
3. `int␣x;xx=8;`

Constructing a CFG to Solve a Problem

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

$$001011010110 = 0\bullet\,\mathbf{0101}\,\bullet 1\bullet\,\mathbf{010110}$$

(↑ in $\mathcal{L}_{\text{bal}}$; ↑ in $\mathcal{L}_{\text{bal}}$)

$$= 0S1S$$ (S represents "a string in $\mathcal{L}_{\text{bal}}$")

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:

1. Every string generated by this CFG is in $\mathcal{L}_{\text{bal}}$?
2. 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 CFG_{bal} *generates a string in* $\mathcal{L}_{bal}$.

Strong induction on the length of the derivation (number of production rules invoked).
Base Case. length-1 derivation gives ε.
Induction. The derivation starts in one of two ways:

$$S \rightarrow 0\,\underset{x}{S}\,1\,\underset{y}{S} \rightarrow \cdots \qquad \text{or} \qquad S \rightarrow 1\,\underset{x}{S}\,0\,\underset{y}{S} \rightarrow \cdots$$

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 $\mathcal{L}_{bal}$ *can be derived within* CFG_{bal}.

Strong induction on the length of the string.
Base case: length-1 string, ε.
Induction. *Any* string w in $\mathcal{L}_{\text{bal}}$ has one of two forms:

$$w = 0w_1 1w_2 \qquad \text{or} \qquad w = 1w_1 0w_2,$$

where $w_1, w_2 \in \mathcal{L}_{\text{bal}}$ and have smaller length.
By the induction hypothesis, $S \overset{*}{\Rightarrow} w_1$ and $S \overset{*}{\Rightarrow} w_2$, so $S \overset{*}{\Rightarrow} w$.

Practice. Exercise 25.5.

Union, Concatenation, Kleene-star

$\mathcal{L}_1 = \{0^{\bullet n}1^{\bullet n} \mid n \geq 0\}$
$A \rightarrow \varepsilon \mid 0A1$

$\mathcal{L}_2 = \{1^{\bullet n}0^{\bullet n} \mid n \geq 0\}$
$B \rightarrow \varepsilon \mid 1B0$

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

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

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

1: $S \rightarrow \varepsilon \mid SA$ ← generates $A^{\bullet i}$
2: $A \rightarrow \varepsilon \mid 0A1$ ← each A becomes a $0^{\bullet n}1^{\bullet n}$

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

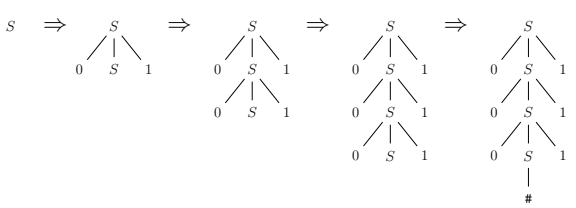
Parse Trees

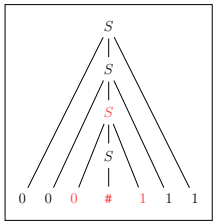
$$S \to \# \mid 0S1$$

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.

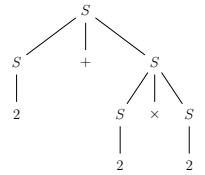
CFG for Arithmetic

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

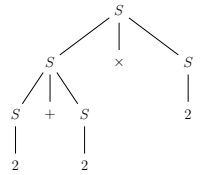
(the terminals are $+$, $\times$, $($, $)$ and 2)

Two derivations of $2 + 2 \times 2$ along with parse trees,

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



(multiply 2×2 and add to 2)



(add $2 + 2$ and multiply by 2)

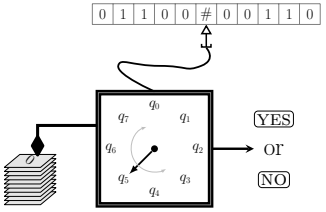
- Parse tree ↔ How you interpret the string.
- Different parse trees ↔ different meanings.
- **BAD!** We want unambiguous meaning
  programs, html-code, math, English, . . .

Unambiguous grammer

$$\begin{aligned} 1{:}\ S &\to P \mid S + P \\ 2{:}\ P &\to T \mid P \times T \\ 3{:}\ T &\to 2 \mid (S) \end{aligned}$$

Pushdown Automata: DFAs with Stack Memory

$$\mathcal{L} = \{w\#w^{\mathrm{R}} \mid w \in \{0,1\}^*\}$$
$$S \to \# \mid 0S0 \mid 1S1$$



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

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

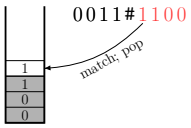
For each bit in the second half, pop the stack and compare.

DFAs with stack memory closely related to CFGs.

Non Context Free

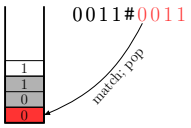
$\{w\#w\}$	repetition
$\{0^{\bullet n}1^{\bullet n}0^{\bullet n}\}$	multiple-equality
$\{0^{\bullet n^2}\}, \{0^{\bullet n}1^{\bullet n^2}\}$	squaring
$\{0^{\bullet 2^n}\}, \{0^{\bullet n}1^{\bullet 2^n}\}$	exponentiation

$w\#w^{\mathrm{R}}$



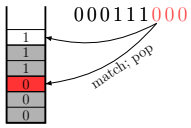
0011 is pushed.
DFA matches 1100 by popping.

$w\#w$



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

$0^{\bullet n}1^{\bullet n}0^{\bullet n}$



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).