Foundations of Computer Science
Lecture 25

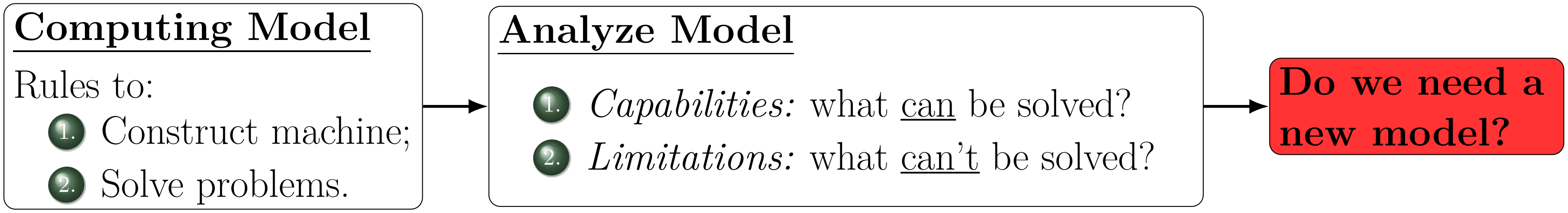
Context Free Grammars (CFGs)

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 <u>can</u> be solved?
2. *Limitations:* what <u>can't</u> be solved?

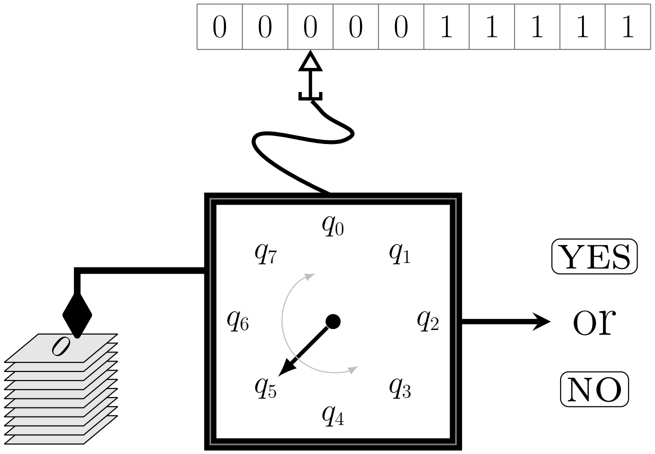
Do we need a new model?

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 $\{0^{\bullet n}1^{\bullet n} \mid n \geq 0\}$?

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. $x \in \mathcal{L}_{0^n1^n} \rightarrow 0 \bullet x \bullet 1 \in \mathcal{L}_{0^n1^n}$. **[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:

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:

ε

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:

$$\varepsilon \quad \to \quad 01$$

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. $x \in \mathcal{L}_{0^n1^n} \rightarrow 0 \bullet x \bullet 1 \in \mathcal{L}_{0^n1^n}$. **[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:

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

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. $x \in \mathcal{L}_{0^n1^n} \rightarrow 0 \bullet x \bullet 1 \in \mathcal{L}_{0^n1^n}$. **[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:

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

$\boxed{\text{NO}}$

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:

$$\varepsilon \quad \to \quad 01 \quad \to \quad 0011 \quad \to \quad 000111$$

$\boxed{\text{NO}}$

A Context Free Grammar is like a recursive definition.

$$1:\quad S \to \varepsilon \qquad\qquad \left(\varepsilon \in \mathcal{L}_{0^n1^n}\right)$$

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:

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

$\boxed{\text{NO}}$

A Context Free Grammar is like a recursive definition.

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

Rules for Context Free Grammars (CFGs)

Production rules of the CFG:

$$\begin{aligned} \texttt{1:}\quad S &\to \varepsilon \\ \texttt{2:}\quad S &\to 0S1 \end{aligned}$$

Each production rule has the form

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

Rules for Context Free Grammars (CFGs)

Production rules of the CFG:

1: $S \rightarrow \varepsilon$

2: $S \rightarrow 0S1$

Each production rule has the form

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

S

1: Write down the start variable (form the first production rule, typically S).

Rules for Context Free Grammars (CFGs)

Production rules of the CFG:

$$\begin{aligned} 1\!: \quad S &\to \varepsilon \\ 2\!: \quad S &\to 0S1 \end{aligned}$$

Each production rule has the form

$$\begin{tikzcd} \texttt{variable} \arrow[r] & \texttt{expression} \\ P, Q, R, S, T, \ldots \arrow[u] & \text{string of variables and terminals} \arrow[u] \end{tikzcd}$$

$$\begin{tikzcd} S \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 0S1 \\ \varepsilon \end{tikzcd}$$

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.

"Replace `variable` with `expression`, no matter where (independent of context)"

Rules for Context Free Grammars (CFGs)

Production rules of the CFG:

$$\begin{array}{rlcl} \texttt{1:} & S & \rightarrow & \varepsilon \\ \texttt{2:} & S & \rightarrow & 0S1 \end{array}$$

Each production rule has the form

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

$$\begin{tikzcd} S \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 0S1 \\ \varepsilon & \end{tikzcd}$$

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)"

Rules for Context Free Grammars (CFGs)

Production rules of the CFG:

1: $S \rightarrow \varepsilon$

2: $S \rightarrow 0S1$

Each production rule has the form

`variable` → `expression`

↑ ↑

$P, Q, R, S, T, \ldots$ string of variables and terminals

$$\begin{tikzcd}
S \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 0S1 \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 00S11 \\
\varepsilon & 01 &
\end{tikzcd}$$

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)”

Rules for Context Free Grammars (CFGs)

Production rules of the CFG:

$$\begin{aligned} &\text{1:}\quad S \;\to\; \varepsilon \\ &\text{2:}\quad S \;\to\; 0S1 \end{aligned}$$

Each production rule has the form

$$\begin{tikzcd} \texttt{variable} \arrow[r, phantom, "\to"] & \texttt{expression} \\ P, Q, R, S, T, \ldots \arrow[u, "\uparrow" description, phantom] & \text{string of variables and terminals} \arrow[u, "\uparrow" description, phantom] \end{tikzcd}$$

$$\begin{tikzcd} S \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 0S1 \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 00S11 \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 000S111 \\ \varepsilon & 01 & 0011 & \end{tikzcd}$$

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)"

Rules for Context Free Grammars (CFGs)

Production rules of the CFG:

$$\begin{aligned} 1:&\quad S \to \varepsilon \\ 2:&\quad S \to 0S1 \end{aligned}$$

Each production rule has the form

`variable` → `expression`

↑ ↑

$P, Q, R, S, T, \ldots$ string of variables and terminals

$$\begin{tikzcd}
S \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 0S1 \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 00S11 \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 000S111 \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 0000S1111 \\
\varepsilon & 01 & 0011 & \color{red}{000111} &
\end{tikzcd}$$

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)”

Rules for Context Free Grammars (CFGs)

Production rules of the CFG:

$$\begin{aligned} {\scriptstyle 1:}\quad S &\to \varepsilon \\ {\scriptstyle 2:}\quad S &\to 0S1 \end{aligned}$$

Each production rule has the form

$$\begin{tikzcd} \texttt{variable} & \to & \texttt{expression} \\ P, Q, R, S, T, \ldots \arrow[u] & & \text{string of variables and terminals} \arrow[u] \end{tikzcd}$$

$$\begin{tikzcd} S \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 0S1 \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 00S11 \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 000S111 \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & 0000S1111 \arrow[r, Rightarrow, "2:"] \arrow[d, Rightarrow, "1:"] & \cdots \\ \varepsilon & 01 & 0011 & 000111 & 00001111 & \end{tikzcd}$$

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

Language of Equality, CFG_{bal}

CFG_{bal}

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

Language of Equality, CFG$_{\text{bal}}$

CFG$_{\text{bal}}$ $\quad\quad$ 1: $S \rightarrow \varepsilon \mid 0S1S \mid 1S0S$

A *derivation* of 0110 in CFG$_{\text{bal}}$ (each step is called an inference).

$$S \overset{1:}{\Rightarrow} 0S1S$$

Language of Equality, $\textsc{CFG}_{\text{bal}}$

$$\textsc{CFG}_{\text{bal}} \qquad\qquad 1{:}\quad S \;\to\; \varepsilon \mid 0S1S \mid 1S0S$$

A *derivation* of 0110 in $\textsc{CFG}_{\text{bal}}$ (each step is called an inference).

$$S \overset{1;}{\Rightarrow} 0S1S \overset{1;}{\Rightarrow} 0S11S0S$$

Language of Equality, CFG$_{\text{bal}}$

CFG$_{\text{bal}}$

$$\text{1:}\quad S \;\rightarrow\; \varepsilon \mid 0S1S \mid 1S0S$$

A *derivation* of 0110 in CFG$_{\text{bal}}$ (each step is called an inference).

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

Language of Equality, CFG$_{\text{bal}}$

CFG$_{\text{bal}}$ $\quad$ 1: $S \rightarrow \varepsilon \mid 0S1S \mid 1S0S$

A *derivation* of 0110 in CFG$_{\text{bal}}$ (each step is called an inference).

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

Notation:

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

Language of Equality, CFG_{bal}

$$\textbf{CFG}_{\text{bal}} \qquad\qquad 1: \; S \to \varepsilon \mid 0S1S \mid 1S0S$$

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),

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

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

Language of Equality, $\textsc{cfg}_{\text{bal}}$

$\textsc{cfg}_{\text{bal}}$ $\quad$ 1: $S \rightarrow \varepsilon \mid 0S1S \mid 1S0S$

A *derivation* of 0110 in $\textsc{cfg}_{\text{bal}}$ (each step is called an inference).

$$S \overset{1:}{\Rightarrow} 0S1S \overset{1:}{\Rightarrow} 0S11S0S \overset{1:}{\Rightarrow} 0\varepsilon 11S0S \overset{*}{\Rightarrow} 0110$$

Notation:

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

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

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

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.

ⓐ 0011
ⓑ 0110
ⓒ 00011
ⓓ 010101

Give an informal description for the CFL of this CFG.

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: $T_1 \rightarrow 1$

A CFG for English

1: S → `<phrase><verb>`
2: `<phrase>` → `<article><noun>`
3: `<article>` → A␣ | The␣
4: `<noun>` → cat␣ | dog␣
5: `<verb>` → walks. | runs. | walks.␣S | runs.␣S

A CFG for English

1: $S \rightarrow$ `<phrase><verb>`

2: `<phrase>` $\rightarrow$ `<article><noun>`

3: `<article>` $\rightarrow$ A␣ | The␣

4: `<noun>` $\rightarrow$ cat␣ | dog␣

5: `<verb>` $\rightarrow$ walks. | runs. | walks.␣S | runs.␣S

$S \overset{1:}{\Rightarrow}$ `<phrase><verb>`

A CFG for English

$$
\begin{aligned}
\texttt{1:}\quad & S & \rightarrow\ & \texttt{<phrase><verb>} \\
\texttt{2:}\quad & \texttt{<phrase>} & \rightarrow\ & \texttt{<article><noun>} \\
\texttt{3:}\quad & \texttt{<article>} & \rightarrow\ & \text{A}\textvisiblespace \mid \text{The}\textvisiblespace \\
\texttt{4:}\quad & \texttt{<noun>} & \rightarrow\ & \text{cat}\textvisiblespace \mid \text{dog}\textvisiblespace \\
\texttt{5:}\quad & \texttt{<verb>} & \rightarrow\ & \text{walks.} \mid \text{runs.} \mid \text{walks.}\textvisiblespace S \mid \text{runs.}\textvisiblespace S
\end{aligned}
$$

$$
\begin{aligned}
S & \overset{1:}{\Rightarrow} \texttt{<phrase><verb>} \\
& \overset{5:}{\Rightarrow} \texttt{<phrase>}\text{walks.}
\end{aligned}
$$

A CFG for English

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

$$
\begin{array}{rcl}
S & \overset{1:}{\Rightarrow} & \texttt{<phrase><verb>} \\
 & \overset{5:}{\Rightarrow} & \texttt{<phrase>}\text{walks.} \\
 & \overset{2:}{\Rightarrow} & \texttt{<article><noun>}\text{walks.}
\end{array}
$$

A CFG for English

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

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

A CFG for English

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

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

Pop Quiz. Give a derivation for: A␣cat␣runs.␣The␣dog␣walks.

A CFG for Programming

$$
\begin{array}{lrcl}
\text{1:} & S & \rightarrow & \texttt{<stmt>;}S \mid \texttt{<stmt>;} \\
\text{2:} & \texttt{<stmt>} & \rightarrow & \texttt{<assign>} \mid \texttt{<declare>} \\
\text{3:} & \texttt{<declare>} & \rightarrow & \texttt{int}\textvisiblespace\texttt{<variable>} \\
\text{4:} & \texttt{<assign>} & \rightarrow & \texttt{<variable>=<integer>} \\
\text{5:} & \texttt{<integer>} & \rightarrow & \texttt{<integer><digit>} \mid \texttt{<digit>} \\
\text{6:} & \texttt{<digit>} & \rightarrow & \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\text{7:} & \texttt{<variable>} & \rightarrow & \texttt{x} \mid \texttt{x<variable>}
\end{array}
$$

A CFG for Programming

1: $S \rightarrow \texttt{<stmt>;}S \mid \texttt{<stmt>;}$

2: $\texttt{<stmt>} \rightarrow \texttt{<assign>} \mid \texttt{<declare>}$

3: $\texttt{<declare>} \rightarrow \texttt{int␣<variable>}$

4: $\texttt{<assign>} \rightarrow \texttt{<variable>=<integer>}$

5: $\texttt{<integer>} \rightarrow \texttt{<integer><digit>} \mid \texttt{<digit>}$

6: $\texttt{<digit>} \rightarrow \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9}$

7: $\texttt{<variable>} \rightarrow \texttt{x} \mid \texttt{x<variable>}$

Pop Quiz. Give derivations for these snippets of code.

ⓐ `int␣x;int␣xx;x=22;xx=8;`

ⓑ `x=8;int␣x;`

ⓒ `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

Constructing a CFG to Solve a Problem

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

$$001011010110 \;=\; 0 \qquad\qquad\qquad 1$$

Constructing a CFG to Solve a Problem

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

$$\begin{aligned} 001011010110 &= 0\bullet\ \underset{\substack{\uparrow \\ \text{in } \mathcal{L}_{\text{bal}}}}{\mathbf{0101}}\ \bullet 1 \bullet\ \underset{\substack{\uparrow \\ \text{in } \mathcal{L}_{\text{bal}}}}{\mathbf{010110}} \\ &= 0S1S \end{aligned}$$

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

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

Constructing a CFG to Solve a Problem

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

$$\begin{aligned} 001011010110 &= 0\bullet \underset{\underset{\text{in } \mathcal{L}_{\text{bal}}}{\uparrow}}{\mathbf{0101}} \bullet 1 \bullet \underset{\underset{\text{in } \mathcal{L}_{\text{bal}}}{\uparrow}}{\mathbf{010110}} \\ &= 0S1S \end{aligned}$$

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

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

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

Constructing a CFG to Solve a Problem

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

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

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

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

We must *prove* that:

- (i) Every string generated by this CFG is in $\mathcal{L}_{\text{bal}}$?
- (ii) 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* $\mathcal{L}_{bal}$.

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

Proving a CFG Solves a Problem

$$\begin{aligned}\mathcal{L}_{\text{bal}} &= \{\text{strings with an equal number of 1's and 0's}\}\\ S &\to \varepsilon \mid 0S1S \mid 1S0S\end{aligned}$$

(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.

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

Proving a CFG Solves a Problem

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

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

(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:

$$S \to 0 \underset{x}{\underset{\Downarrow *}{S}}\ 1\ \underset{y}{\underset{\Downarrow *}{S}} \to \cdots \qquad \text{or} \qquad S \to 1 \underset{x}{\underset{\Downarrow *}{S}}\ 0\ \underset{y}{\underset{\Downarrow *}{S}} \to \cdots$$

The derivations of x and y are shorter.

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

Proving a CFG Solves a Problem

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

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

(i) Every derivation in $\textsc{cfg}_{bal}$ *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:

$$S \to 0\ \underset{\substack{\Downarrow * \\ x}}{S}\ 1\ \underset{\substack{\Downarrow * \\ y}}{S}\ \to \cdots \qquad\qquad \text{or} \qquad\qquad S \to 1\ \underset{\substack{\Downarrow * \\ x}}{S}\ 0\ \underset{\substack{\Downarrow * \\ y}}{S}\ \to \cdots$$

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* $\textsc{cfg}_{bal}$*.*

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:

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

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.

Proving a CFG Solves a Problem

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

$$S \;\to\; \varepsilon \mid 0S1S \mid 1S0S$$

(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:

$$S \to 0\ \underset{x}{\underset{\Downarrow *}{S}}\ 1\ \underset{y}{\underset{\Downarrow *}{S}}\ \to \cdots \qquad \text{or} \qquad S \to 1\ \underset{x}{\underset{\Downarrow *}{S}}\ 0\ \underset{y}{\underset{\Downarrow *}{S}}\ \to \cdots$$

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.

Proving a CFG Solves a Problem

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

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

(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:

$$S \to 0\ \underset{x}{\underset{\Downarrow *}{S}}\ 1\ \underset{y}{\underset{\Downarrow *}{S}}\ \to \cdots \qquad \text{or} \qquad S \to 1\ \underset{x}{\underset{\Downarrow *}{S}}\ 0\ \underset{y}{\underset{\Downarrow *}{S}}\ \to \cdots$$

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

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

$A \to \varepsilon \mid 0A1$

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

$B \to \varepsilon \mid 1B0$

Union, Concatenation, Kleene-star

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

$A \to \varepsilon \mid 0A1$

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

$B \to \varepsilon \mid 1B0$

$\mathcal{L}_1 \cup \mathcal{L}_2 :$ 1: $S \to A \mid B$

Union, Concatenation, Kleene-star

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

$A \to \varepsilon \mid 0A1$

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

$B \to \varepsilon \mid 1B0$

$\mathcal{L}_1 \cup \mathcal{L}_2$:

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

Union, Concatenation, Kleene-star

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

$A \to \varepsilon \mid 0A1$

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

$B \to \varepsilon \mid 1B0$

$\mathcal{L}_1 \cup \mathcal{L}_2 :$

$$\begin{array}{rlcl} \text{1:} & S & \to & A \mid B \\ \text{2:} & A & \to & \varepsilon \mid 0A1 \\ \text{3:} & B & \to & \varepsilon \mid 1B0 \end{array}$$

$\mathcal{L}_1 \bullet \mathcal{L}_2 :$

Union, Concatenation, Kleene-star

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

$$A \to \varepsilon \mid 0A1$$

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

$$B \to \varepsilon \mid 1B0$$

$\mathcal{L}_1 \cup \mathcal{L}_2$:

$$\begin{array}{rlcl} \text{1:} & S & \to & A \mid B \\ \text{2:} & A & \to & \varepsilon \mid 0A1 \\ \text{3:} & B & \to & \varepsilon \mid 1B0 \end{array}$$

$\mathcal{L}_1 \bullet \mathcal{L}_2$:

$$\begin{array}{rlcl} \text{2:} & A & \to & \varepsilon \mid 0A1 \\ \text{3:} & B & \to & \varepsilon \mid 1B0 \end{array}$$

Union, Concatenation, Kleene-star

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

$$A \to \varepsilon \mid 0A1$$

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

$$B \to \varepsilon \mid 1B0$$

$\mathcal{L}_1 \cup \mathcal{L}_2$:

$$\begin{array}{lccl} \texttt{1:} & S & \to & A \mid B \\ \texttt{2:} & A & \to & \varepsilon \mid 0A1 \\ \texttt{3:} & B & \to & \varepsilon \mid 1B0 \end{array}$$

$\mathcal{L}_1 \bullet \mathcal{L}_2$:

$$\begin{array}{lccl} \texttt{2:} & A & \to & \varepsilon \mid 0A1 \\ \texttt{3:} & B & \to & \varepsilon \mid 1B0 \end{array}$$

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

1: $S \to \varepsilon \mid SA$ ← generates $A^{\bullet i}$

2: $A \to \varepsilon \mid 0A1$ ← each A becomes a $0^{\bullet n}1^{\bullet n}$

Union, Concatenation, Kleene-star

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

$$A \to \varepsilon \mid 0A1$$

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

$$B \to \varepsilon \mid 1B0$$

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

$$\mathcal{L}_1 \bullet \mathcal{L}_2 : \quad \begin{array}{rlcl} & & & \\ \text{2:} & A & \to & \varepsilon \mid 0A1 \\ \text{3:} & B & \to & \varepsilon \mid 1B0 \end{array}$$

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

$$\begin{array}{rl} \text{1:} & S \to \varepsilon \mid SA \\ \text{2:} & A \to \varepsilon \mid 0A1 \end{array}$$

← generates $A^{\bullet i}$

← each A becomes a $0^{\bullet n}1^{\bullet n}$

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

Union, Concatenation, Kleene-star

$$\mathcal{L}_1 = \{0^{\bullet n}1^{\bullet n} \mid n \geq 0\} \qquad \mathcal{L}_2 = \{1^{\bullet n}0^{\bullet n} \mid n \geq 0\}$$

$$A \rightarrow \varepsilon \mid 0A1 \qquad B \rightarrow \varepsilon \mid 1B0$$

$\mathcal{L}_1 \cup \mathcal{L}_2$:

$$\begin{array}{llll} \texttt{1:} & S & \rightarrow & A \mid B \\ \texttt{2:} & A & \rightarrow & \varepsilon \mid 0A1 \\ \texttt{3:} & B & \rightarrow & \varepsilon \mid 1B0 \end{array}$$

$\mathcal{L}_1 \bullet \mathcal{L}_2$:

$$\begin{array}{llll} \texttt{2:} & A & \rightarrow & \varepsilon \mid 0A1 \\ \texttt{3:} & B & \rightarrow & \varepsilon \mid 1B0 \end{array}$$

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

$\texttt{1:}\, S \rightarrow \varepsilon \mid SA$ ← generates $A^{\bullet i}$

$\texttt{2:}\, A \rightarrow \varepsilon \mid 0A1$ ← each A becomes a $0^{\bullet n}1^{\bullet n}$

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

Union, Concatenation, Kleene-star

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

$A \to \varepsilon \mid 0A1$

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

$B \to \varepsilon \mid 1B0$

$\mathcal{L}_1 \cup \mathcal{L}_2$:

$$\begin{array}{llll} \texttt{1:} & S & \to & A \mid B \\ \texttt{2:} & A & \to & \varepsilon \mid 0A1 \\ \texttt{3:} & B & \to & \varepsilon \mid 1B0 \end{array}$$

$\mathcal{L}_1 \bullet \mathcal{L}_2$:

$$\begin{array}{llll} \texttt{1:} & S & \to & AB \\ \texttt{2:} & A & \to & \varepsilon \mid 0A1 \\ \texttt{3:} & B & \to & \varepsilon \mid 1B0 \end{array}$$

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

1: $S \to \varepsilon \mid SA$ ← generates $A^{\bullet i}$

2: $A \to \varepsilon \mid 0A1$ ← 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

$$S \rightarrow \texttt{\#} \mid 0S1$$

Here is a derivation of 000#111,

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

Parse Trees

$$S \to \texttt{\#} \mid 0S1$$

Here is a derivation of $000\texttt{\#}111$,

$$S \;\Rightarrow\; 0S1 \;\Rightarrow\; 00S11 \;\Rightarrow\; 000S111 \;\Rightarrow\; 000\texttt{\#}111$$

The *parse tree* gives more information than a derivation

$S \quad \Rightarrow$

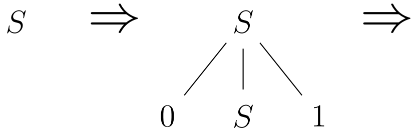
Parse Trees

$$S \rightarrow \texttt{\#} \mid 0S1$$

Here is a derivation of 000**#**111,

$$S \;\Rightarrow\; 0S1 \;\Rightarrow\; 00S11 \;\Rightarrow\; 000S111 \;\Rightarrow\; 000\texttt{\#}111$$

The *parse tree* gives more information than a derivation



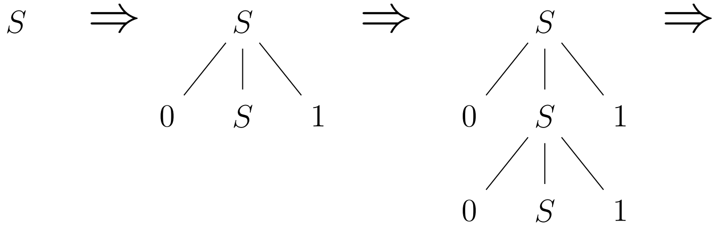
Parse Trees

$$S \rightarrow \texttt{\#} \mid 0S1$$

Here is a derivation of 000#111,

$$S \;\Rightarrow\; 0S1 \;\Rightarrow\; 00S11 \;\Rightarrow\; 000S111 \;\Rightarrow\; 000\texttt{\#}111$$

The *parse tree* gives more information than a derivation



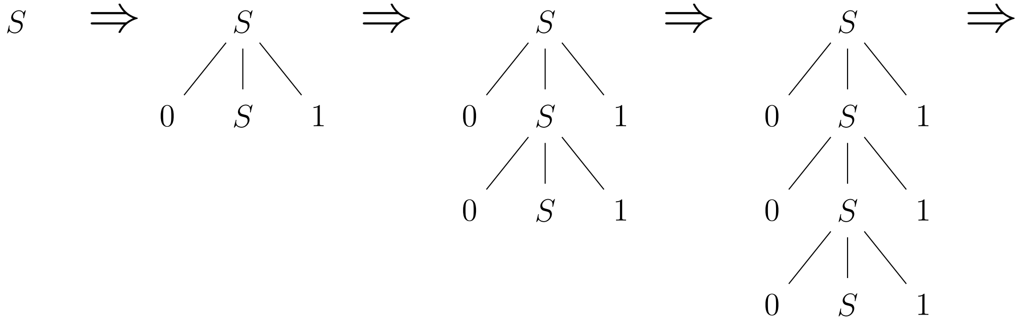
Parse Trees

$$S \to \texttt{\#} \mid 0S1$$

Here is a derivation of 000#111,

$$S \;\Rightarrow\; 0S1 \;\Rightarrow\; 00S11 \;\Rightarrow\; 000S111 \;\Rightarrow\; 000\texttt{\#}111$$

The *parse tree* gives more information than a derivation



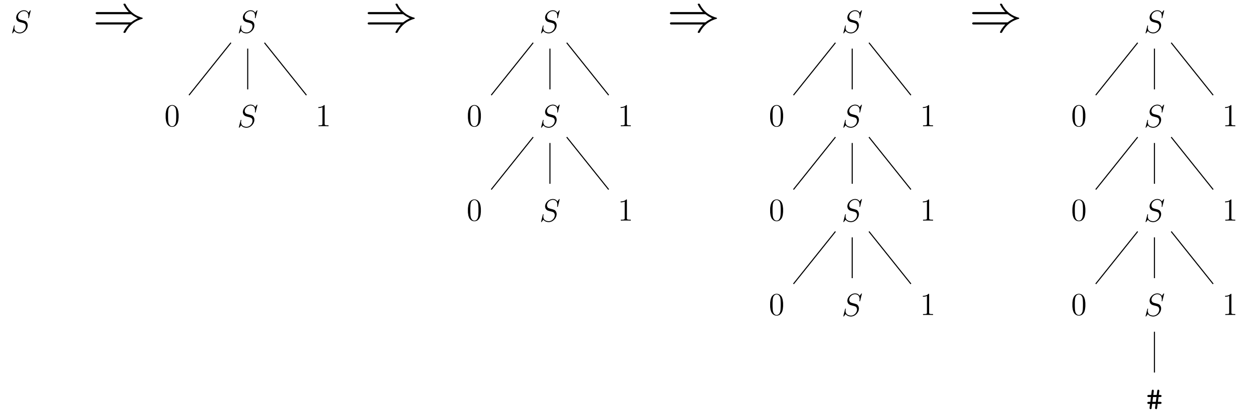
Parse Trees

$$S \rightarrow \texttt{\#} \mid 0S1$$

Here is a derivation of 000#111,

$$S \;\Rightarrow\; 0S1 \;\Rightarrow\; 00S11 \;\Rightarrow\; 000S111 \;\Rightarrow\; 000\texttt{\#}111$$

The *parse tree* gives more information than a derivation



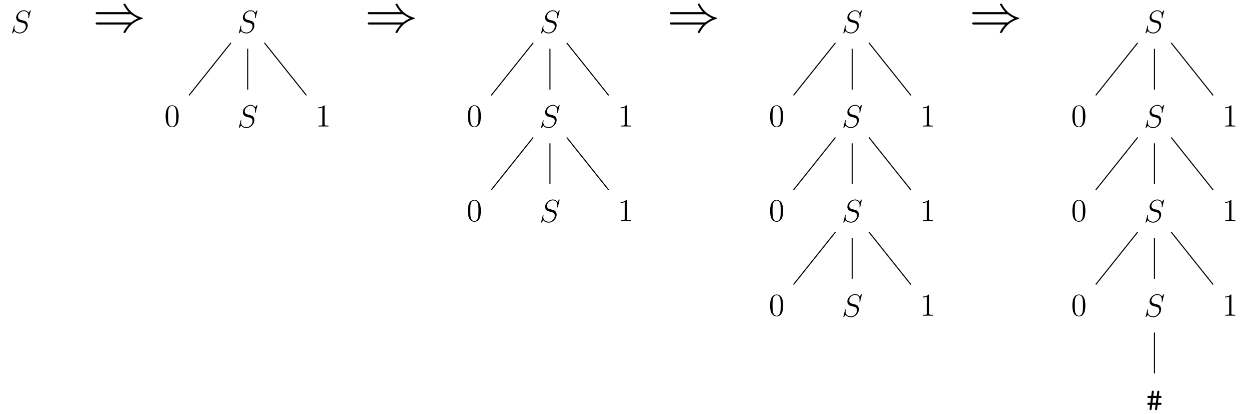
Parse Trees

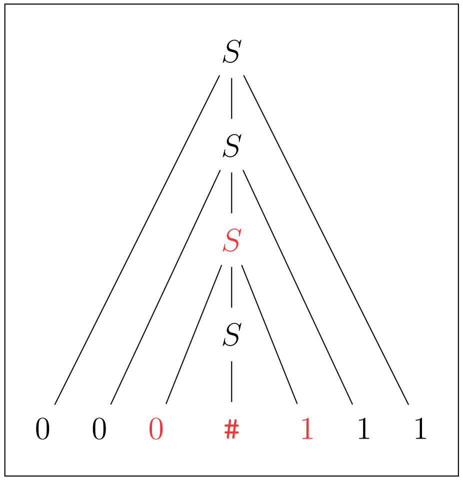
$$S \to \texttt{\#} \mid 0S1$$

Here is a derivation of 000#111,

$$S \;\Rightarrow\; 0S1 \;\Rightarrow\; 00S11 \;\Rightarrow\; 000S111 \;\Rightarrow\; 000\texttt{\#}111$$

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 $+$, $\times$, (,) and 2)

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 $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ S

S $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ S

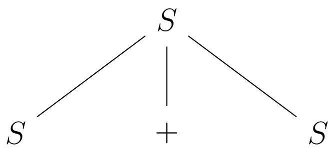
CFG for Arithmetic

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

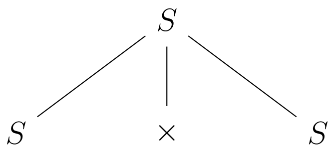
(the terminals are $+$, $\times$, (,) and 2)

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

$S \Rightarrow S + S$



$S \Rightarrow S \times S$



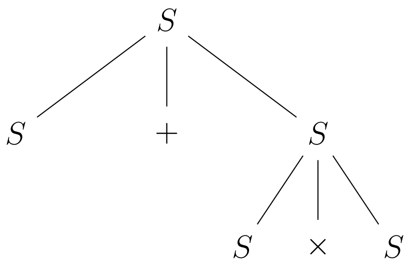
CFG for Arithmetic

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

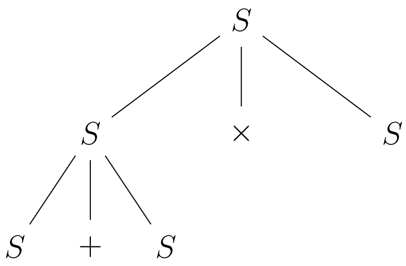
(the terminals are $+$, $\times$, (,) and 2)

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

$$S \Rightarrow S + S \Rightarrow S + S \times S$$



$$S \Rightarrow S \times S \Rightarrow S + S \times S$$



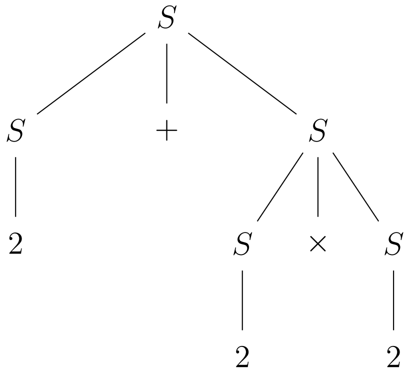
CFG for Arithmetic

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

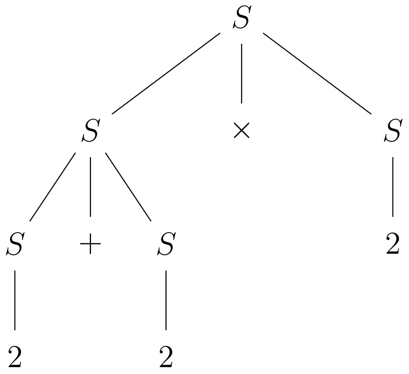
(the terminals are $+$, $\times$, (,) and 2)

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)

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, . . .

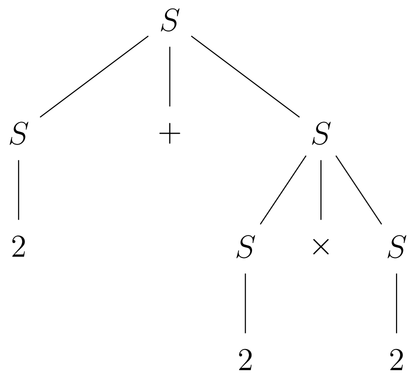
CFG for Arithmetic

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

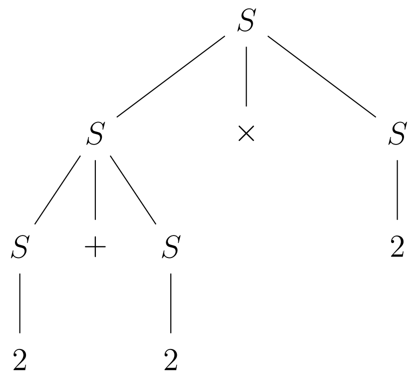
(the terminals are $+$, $\times$, $($, $)$ and 2)

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, …

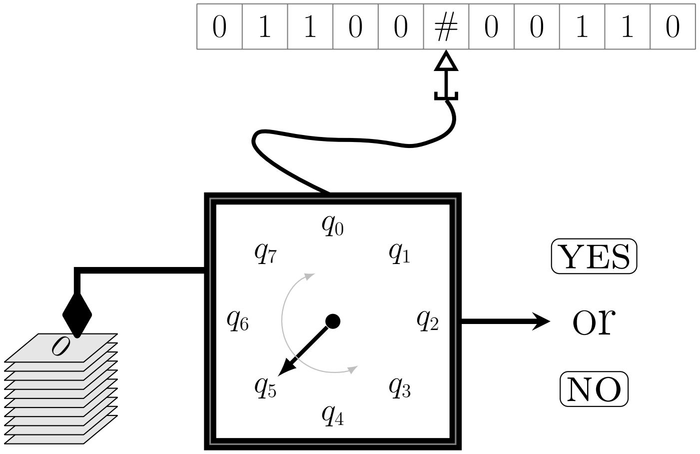
Unambiguous grammer

$$\begin{array}{lccl} 1: & S & \rightarrow & P \mid S + P \\ 2: & P & \rightarrow & T \mid P \times T \\ 3: & T & \rightarrow & 2 \mid (S) \end{array}$$

Pushdown Automata: DFAs with Stack Memory

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

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

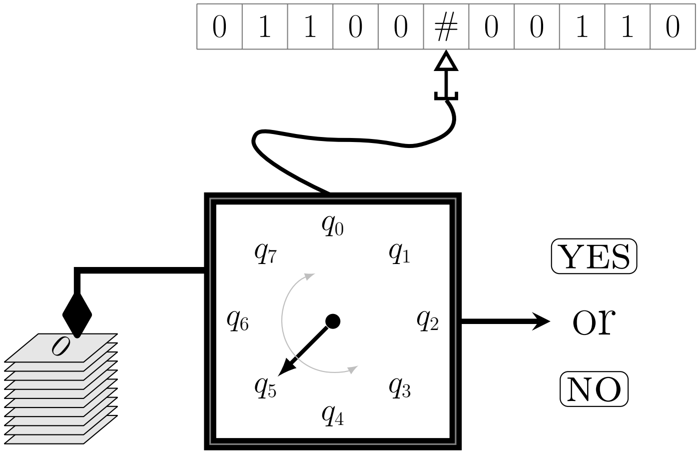


DFA with stack memory (push, pop, read).

Pushdown Automata: DFAs with Stack Memory

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

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



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

$\{w\#w\}$	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

Non Context Free

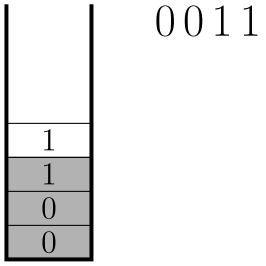
$\{w\#w\}$	repetition
$\{0^{\bullet n}1^{\bullet n}0^{\bullet n}\}$	multiple-equality
$\{0^{\bullet n^2}\}, \{0^{\bullet n}1^{\bullet n^2}\}$	squaring
$\{0^{\bullet 2^n}\}, \{0^{\bullet n}1^{\bullet 2^n}\}$	exponentiation

$\underline{w\#w^{\text{R}}}$ $\qquad$ $\underline{w\#w}$ $\qquad$ $\underline{0^{\bullet n}1^{\bullet n}0^{\bullet n}}$

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

$\underline{w\#w^{\mathrm{R}}}$ $\underline{w\#w}$ $\underline{0^{\bullet n}1^{\bullet n}0^{\bullet n}}$

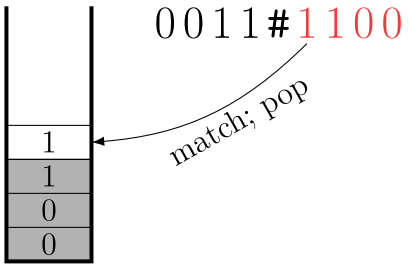


0011 is pushed.

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

$\underline{w\#w^{\text{R}}}$ $\underline{w\#w}$ $\underline{0^{\bullet n}1^{\bullet n}0^{\bullet n}}$

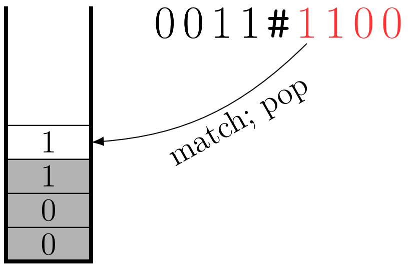


0011 is pushed.
DFA matches 1100 by popping.

Non Context Free

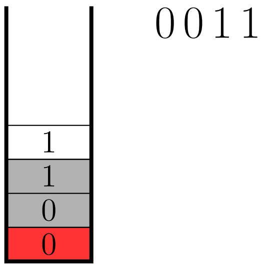
$\{w\#w\}$	repetition
$\{0^{\bullet n}1^{\bullet n}0^{\bullet n}\}$	multiple-equality
$\{0^{\bullet n^2}\}, \{0^{\bullet n}1^{\bullet n^2}\}$	squaring
$\{0^{\bullet 2^n}\}, \{0^{\bullet n}1^{\bullet 2^n}\}$	exponentiation

$\underline{w\#w^{\text{R}}}$



0011 is pushed.
DFA matches 1100 by popping.

$\underline{w\#w}$



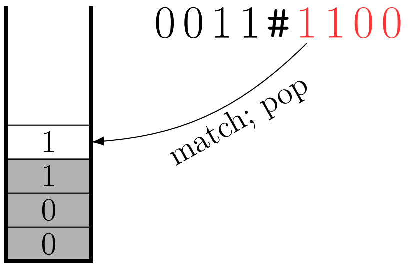
0011 is pushed.

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

Non Context Free

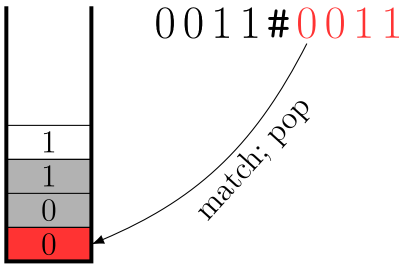
$\{w\#w\}$	repetition
$\{0^{\bullet n}1^{\bullet n}0^{\bullet n}\}$	multiple-equality
$\{0^{\bullet n^2}\}, \{0^{\bullet n}1^{\bullet n^2}\}$	squaring
$\{0^{\bullet 2^n}\}, \{0^{\bullet n}1^{\bullet 2^n}\}$	exponentiation

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



0011 is pushed.
DFA matches 1100 by popping.

$w\#w$



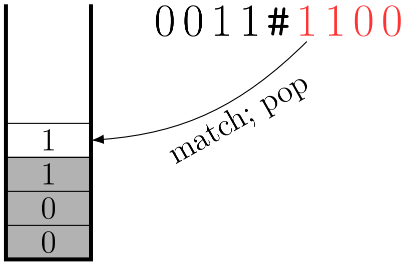
0011 is pushed.
DFA needs bottom-access to match.

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

Non Context Free

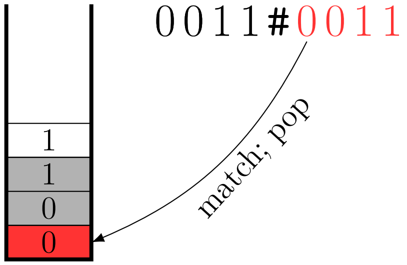
$\{w\#w\}$	repetition
$\{0^{\bullet n}1^{\bullet n}0^{\bullet n}\}$	multiple-equality
$\{0^{\bullet n^2}\}, \{0^{\bullet n}1^{\bullet n^2}\}$	squaring
$\{0^{\bullet 2^n}\}, \{0^{\bullet n}1^{\bullet 2^n}\}$	exponentiation

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



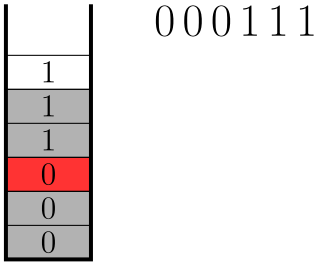
0011 is pushed.
DFA matches 1100 by popping.

$\underline{w\#w}$



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

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

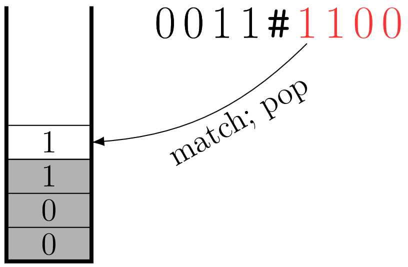


000111 is pushed.

Non Context Free

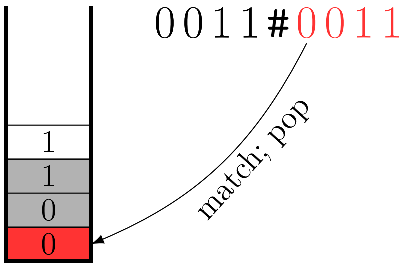
$\{w\#w\}$	repetition
$\{0^{\bullet n}1^{\bullet n}0^{\bullet n}\}$	multiple-equality
$\{0^{\bullet n^2}\}, \{0^{\bullet n}1^{\bullet n^2}\}$	squaring
$\{0^{\bullet 2^n}\}, \{0^{\bullet n}1^{\bullet 2^n}\}$	exponentiation

$\underline{w\#w^{\text{R}}}$



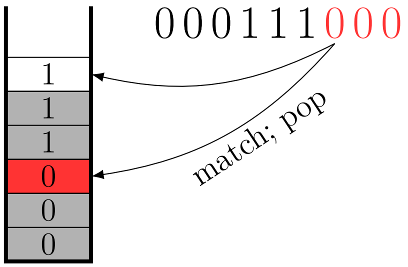
0011 is pushed.
DFA matches 1100 by popping.

$\underline{w\#w}$



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

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

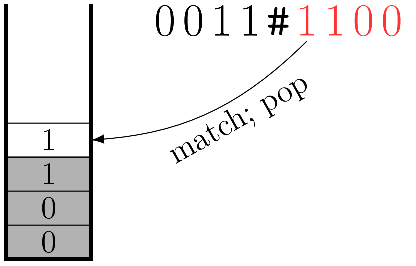


000111 is pushed.
DFA needs random access to match.

Non Context Free

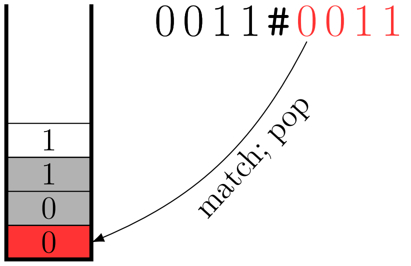
$\{w\#w\}$	repetition
$\{0^{\bullet n}1^{\bullet n}0^{\bullet n}\}$	multiple-equality
$\{0^{\bullet n^2}\}, \{0^{\bullet n}1^{\bullet n^2}\}$	squaring
$\{0^{\bullet 2^n}\}, \{0^{\bullet n}1^{\bullet 2^n}\}$	exponentiation

$\underline{w\#w^{\text{R}}}$



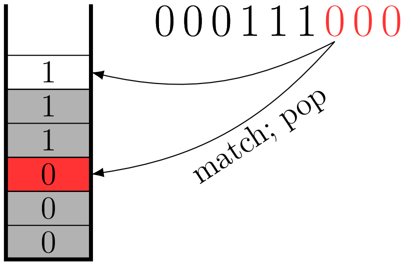
0011 is pushed.
DFA matches 1100 by popping.

$\underline{w\#w}$



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

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



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).