# Context-Free Grammars (and Languages)

Lecture 7

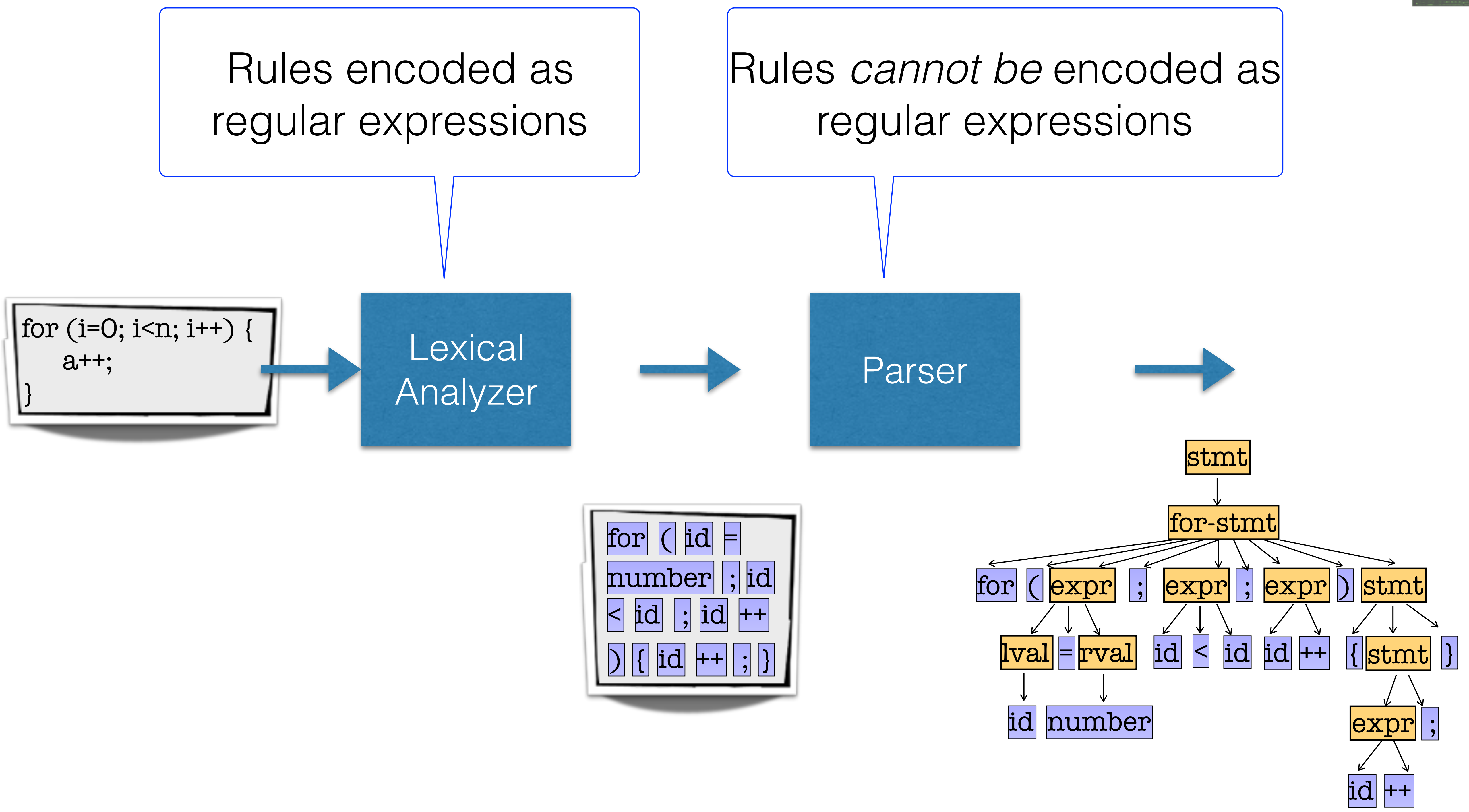
# Today

Beyond regular expressions:
Context-Free Grammars (CFGs)

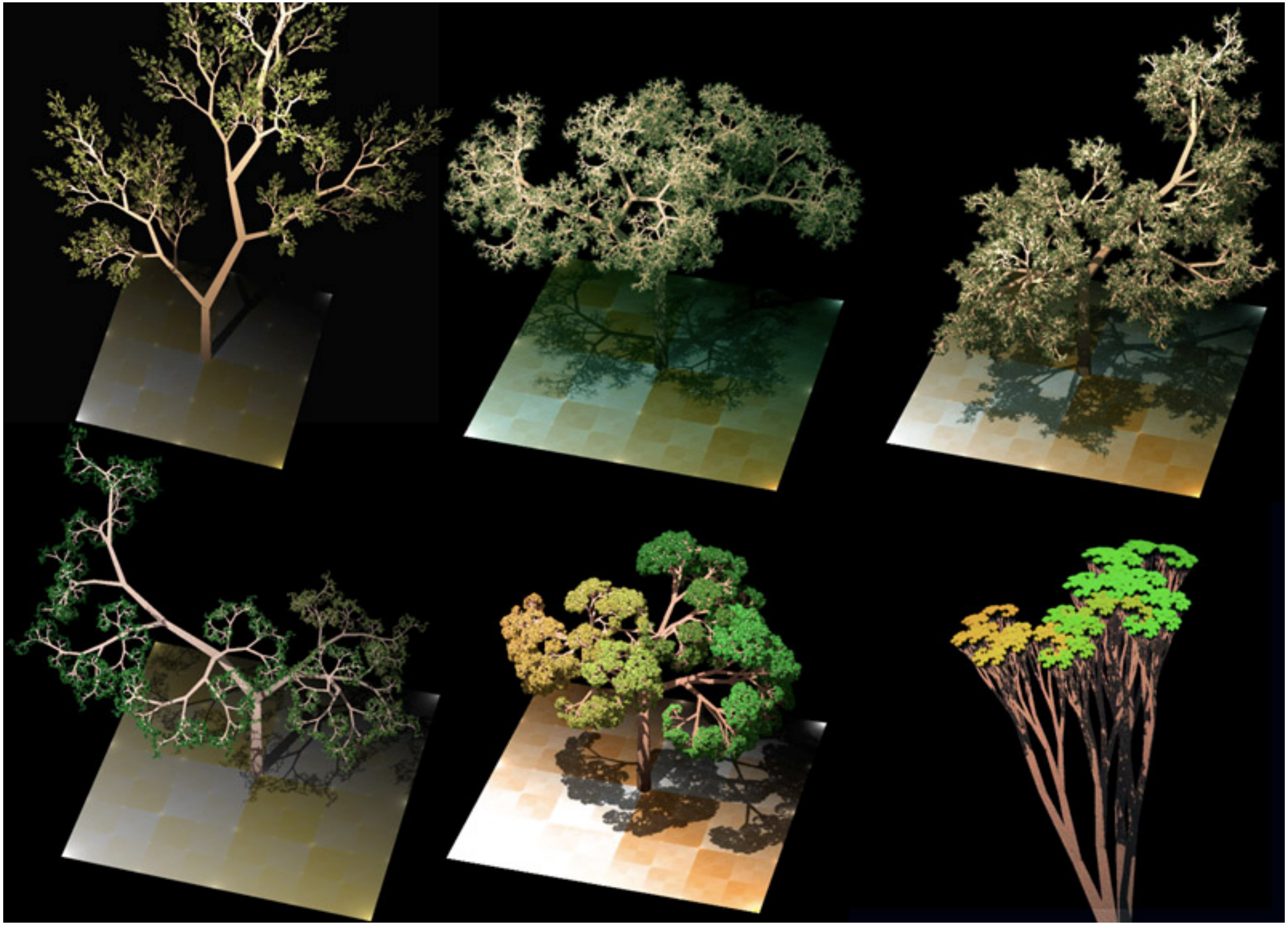
What is a CFG?
What is the language associated with a CFG?

Creating CFGs. Reasoning about CFGs.

# Compiler Frontend

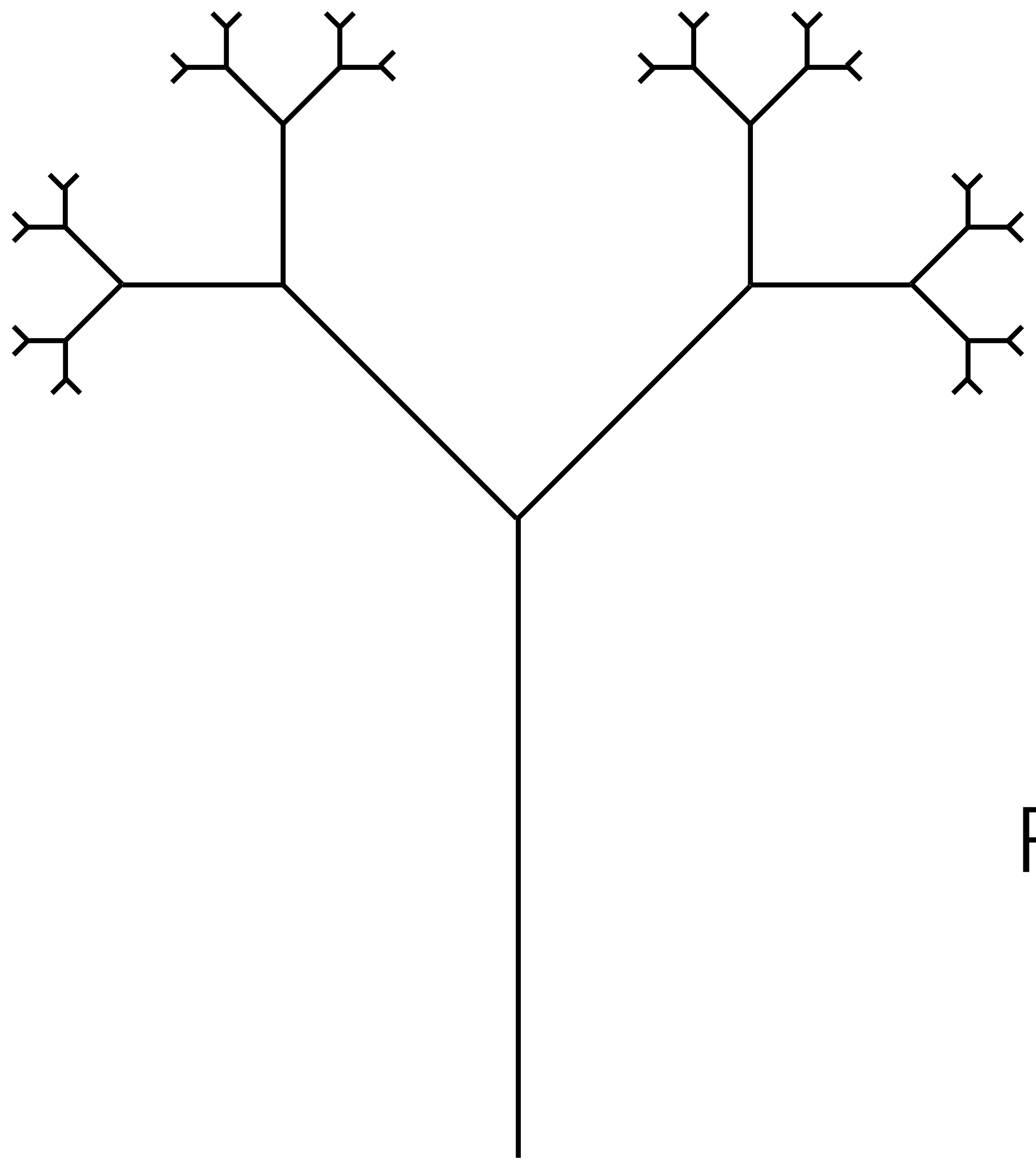


# Biological Models

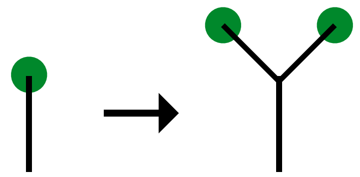


en.wikipedia.org/wiki/**L-system**

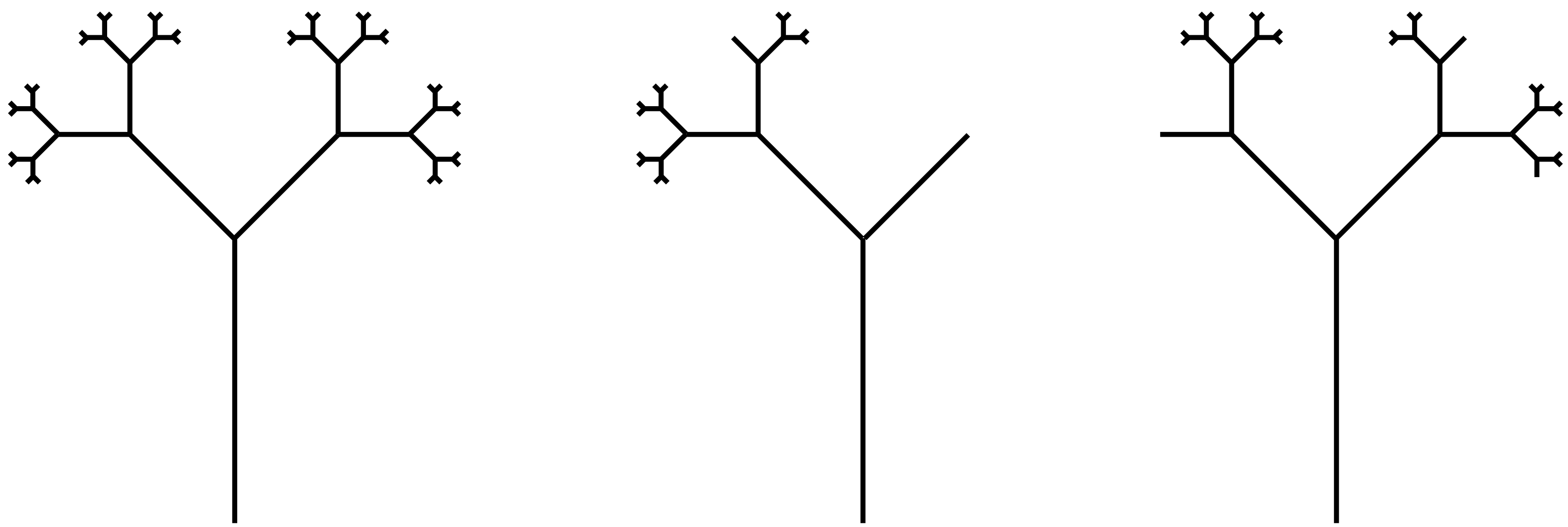
# Biological Models

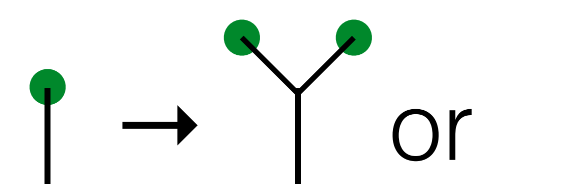


Rule:



# Biological Models



Rule: 


**Grammar**: *Rewriting rules* for generating
a set of strings (i.e., a language) from a "seed"

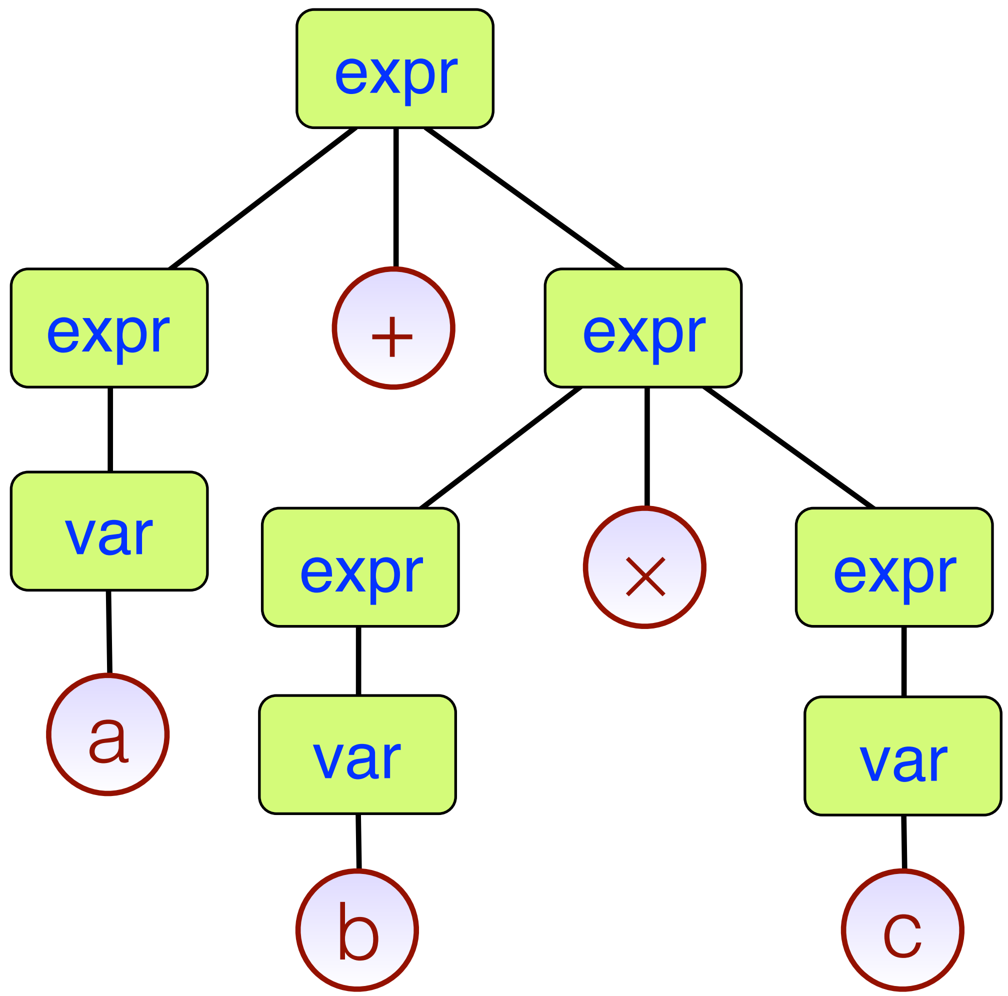
# Context-Free Grammar

Example: a (simplistic) syntax for arithmetic expressions

expr → expr + expr
expr → expr × expr
expr → var
var → a
var → b
var → c

e.g. expr ⇒* a + b × c

"derives"




(This grammar is "ambiguous" since there is another parse tree for the same string)

# Context-Free Grammar

Example: a (simplistic) syntax for arithmetic expressions

expr → expr + expr
expr → expr × expr
expr → var
var → a
var → b
var → c

expr → expr + expr | expr × expr | var
var → a | b | c

short-hand

e.g. expr ⇒* a + b × c

"derives"

$$G = (\Sigma, V, P, S)$$

$\Sigma$ = {a,b,c,+,×} (terminals)

$V$ = {expr, var} (non-terminals)

$P = \{(A,\alpha) \mid A \rightarrow \alpha\}$ (prod. rules)

$S$ = expr (start symbol)

# Context-Free Grammar : Arrows

**Production Rule:** $A \to \pi$, $A \in V$, $\pi \in (\Sigma \cup V)^*$

expr → expr + expr | expr × expr | var
var → a | b | c

**Immediately Derives:** $\alpha_1 \Rightarrow \alpha_2$ if $\alpha_1, \alpha_2 \in (\Sigma \cup V)^*$

s.t., $\alpha_1 = \beta A \gamma$, $\alpha_2 = \beta \pi \gamma$ and $A \to \pi$

More clearly, if grammar is $G$, we write $\alpha \Rightarrow_G^* \alpha'$

expr ⇒ expr + expr
expr + expr ⇒ expr + expr × expr

**Derives:** $\alpha \Rightarrow^* \alpha'$ if $\exists \alpha_1, \ldots, \alpha_{t+1} \in (\Sigma \cup V)^*$ s.t.

$\alpha_1 = \alpha$, $\alpha_{t+1} = \alpha'$, and for all $i \in [1, t]$, $\alpha_i \Rightarrow \alpha_{i+1}$

$t$-step derivation $\alpha \Rightarrow^t \alpha'$

expr $\Rightarrow^*$ expr + expr × expr $\Rightarrow^*$ var + var × var $\Rightarrow^*$ a + b × c

expr $\Rightarrow^*$ a + b × c

# Context-Free Languages

The language ***generated*** by a grammar $G$
with start symbol S and alphabet $\Sigma$,

$$L(G) = \{ w \in \Sigma^* \mid S \Rightarrow_G^* w \}$$

Languages generated by a context free grammars are called **Context Free Languages** (CFL)

# Examples

Over $\Sigma = \{ 0,1 \}$, give a grammar for the following languages:

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

S → ε | 0S1

- $L = \{ w \mid w = w^R \}$

S → ε | 0 | 1 | 0S0 | 1S1

- $L = \{ 0^m 1^n \mid m < n \}$

Z → ε | 0Z1 // $0^n1^n$
S → Z1 | S1 // $0^m1^n$ with m < n

- $L = \{ 0^m 1^n \mid m \neq n \}$

S → A | B
Z → ε | 0Z1 // $0^n1^n$
A → 0Z | 0A // $0^m1^n$ with m > n
B → Z1 | B1 // $0^m1^n$ with m < n

# Parse Tree

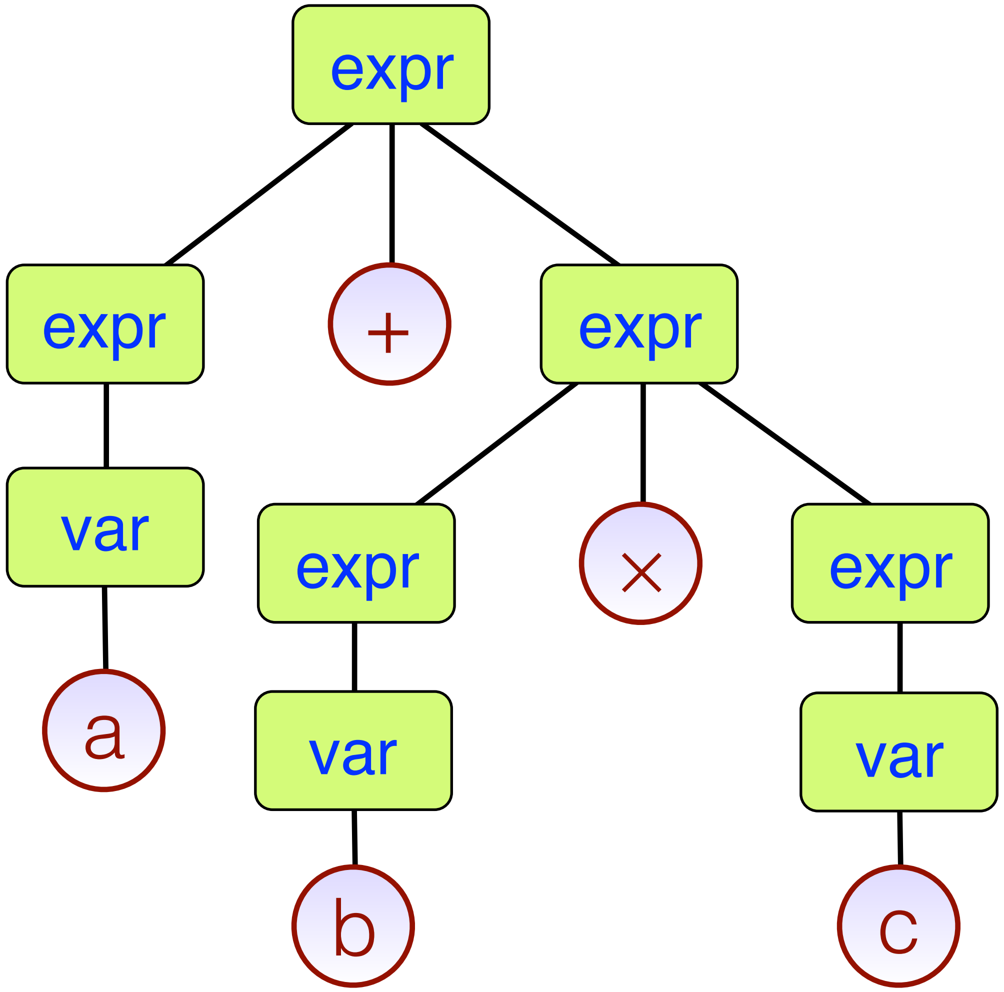
**Parse Tree** captures the structure of derivations for a given string (but not the exact order)

The exact order of derivations is *not* important

But structure is important!

Ambiguous grammar: If some string has two different parse trees

expr ⇒* a + b × c

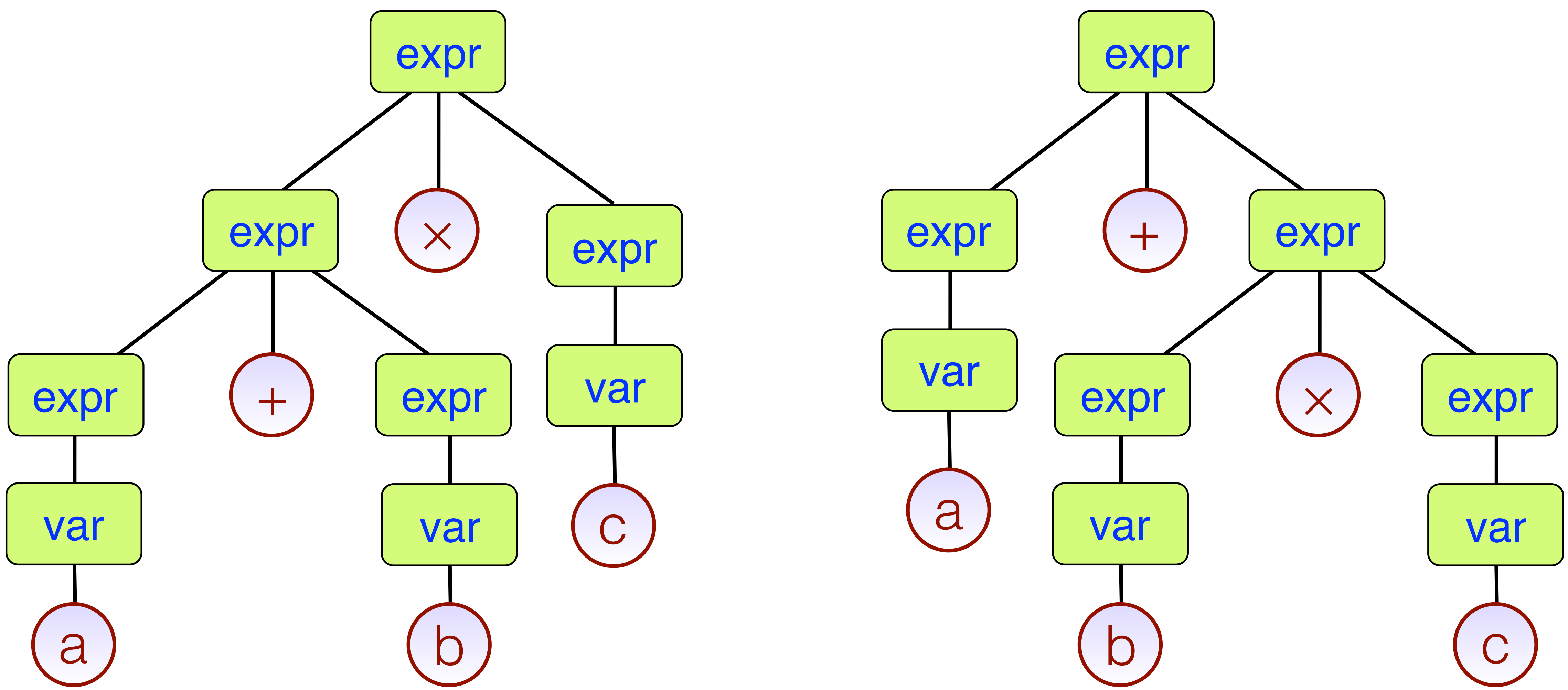



expr ⇒* expr + expr × expr ⇒* var + var × var ⇒* a + b × c
expr ⇒* a + expr ⇒* a + expr × c ⇒* a + b × c

# Ambiguity

expr → expr + expr | expr × expr | var
var → a | b | c

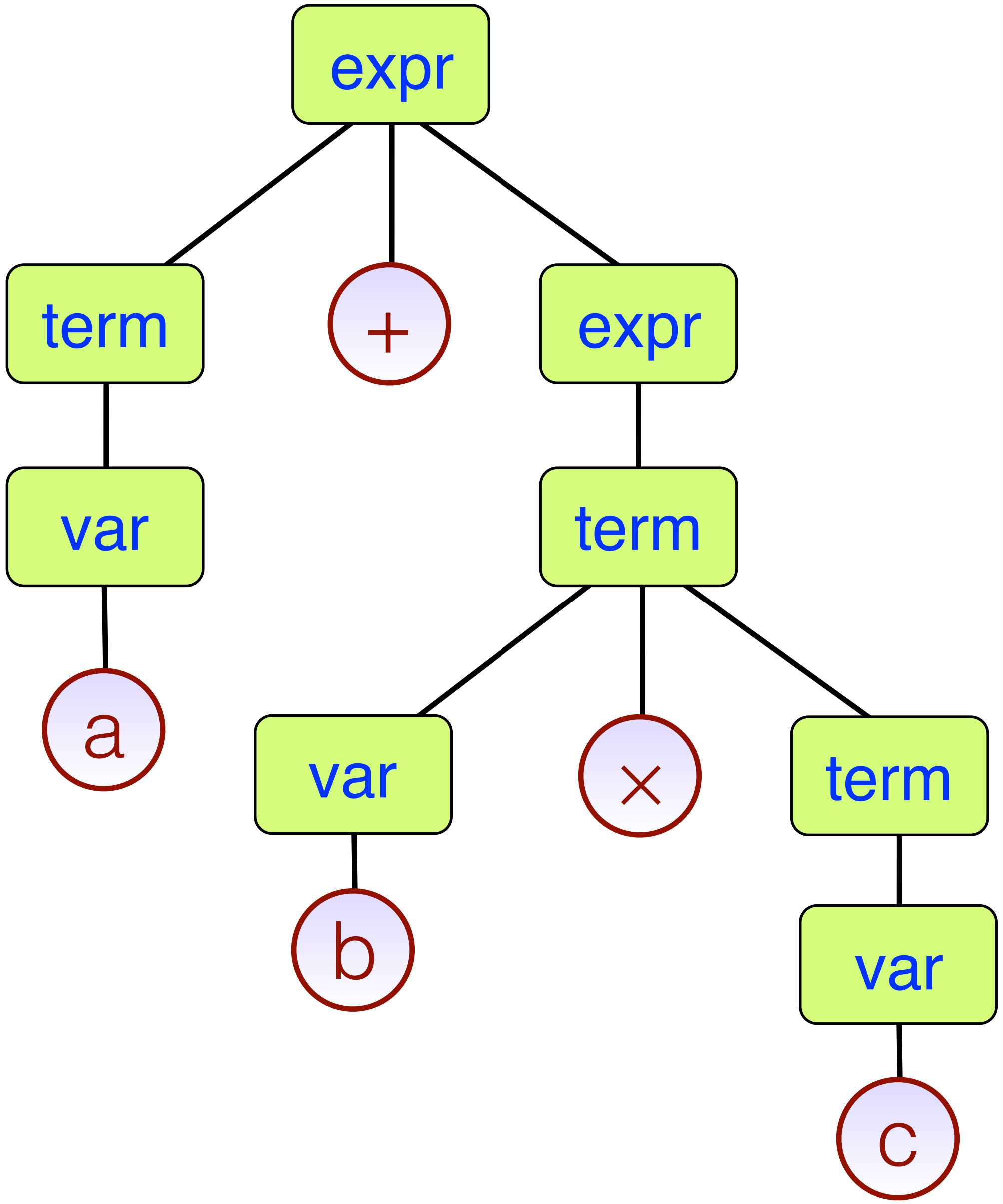
expr ⇒* a + b × c



# An Unambiguous Grammar

expr → term + expr | term
term → var | var × term
var → a | b | c

expr ⇒* a + b × c




In practice, unambiguous grammars are important (e.g., in compilers)

Operator precedence enforced by requiring all × carried out (to get a "term") before any +

There are CFLs which do not have *any* unambiguous grammar:
inherently ambiguous languages

# Examples

- $L = L(0^*)$

$S \rightarrow \varepsilon \mid 0 \mid SS$ : Ambiguous!

$S \rightarrow \varepsilon \mid 0S$ : Unambiguous

- $L$ = set of all strings with balanced parentheses

$S \rightarrow \varepsilon \mid (S) \mid SS$ : Ambiguous!

$T \rightarrow (\,) \mid (S)$
$S \rightarrow \varepsilon \mid TS$ : Unambiguous

# Examples

- $L$ = set of all valid regular expressions over $\{0, 1\}$

An ambiguous grammar (start symbol S, $\Sigma = \{Ø, e, 0, 1, +, *, (, )\}$ ):

S → Ø | $e$ | 0 | 1 | (S) | S* | SS | S+S

An unambiguous grammar for a *subset* of regular expressions:

S → Ø | $e$ | 0 | 1 | (S) | (S*) | (SS) | (S+S)

**Exercise**: An unambiguous grammar for *all* valid regular expressions

# Proving Correctness of Grammars

**Claim:** Let $L = \{ w \mid \#_0(w) = \#_1(w) \}$. Then, $L(G) = L$ where the productions of $G$ are: $S \rightarrow 0S1 \mid 1S0 \mid SS \mid \varepsilon$

**Challenge**: Give an unambiguous grammar

**Proof:** Need to prove both $L(G) \subseteq L$ and $L(G) \supseteq L$.

Prove $L(G) \subseteq L$ by induction on the length of derivations (or height of parse trees)

Prove $L(G) \supseteq L$ by induction on the length of strings.

# Proving Correctness of Grammars

**Claim:** Let $L = \{ w \mid \#_0(w) = \#_1(w) \}$. Then, $L(G) = L$ where the productions of $G$ are: S → $0S1 \mid 1S0 \mid SS \mid \varepsilon$

**Proof:** Proving $L(G) \subseteq L$ by induction on the length of derivations.

Let $w \in L(G)$. $S \Rightarrow^t w$ for some $t \geq 1$. Induction on $t$ to show that $w \in L$.

<u>Base case:</u> $t$=1. Only string derived is $\varepsilon$. ✓

<u>Induction step</u>: Consider $t > 1$. Suppose all $u$ s.t. $S \Rightarrow^k u$, $k < t$, in $L$.

Let $w$ be such that $S \Rightarrow^t w$. i.e., $S \Rightarrow \alpha_1 \Rightarrow^{t\text{-}1} w$.

<u>Case $\alpha_1$=0S1</u>: $w = 0u1$ and $S \Rightarrow^{t\text{-}1} u$. By IH, $\#_0(u)=\#_1(u)$.

Hence $\#_0(w) = \#_0(u)+1 = \#_1(v)+1 = \#_1(w)$. (<u>Case $\alpha_1$=1S0</u> is symmetric.)

<u>Case $\alpha_1$=SS:</u> $w = uv$ and $S \Rightarrow^m u$, $S \Rightarrow^n v$, $1 \leq m,n < t$ $(m+n = t\text{-}1)$. By IH,

$\#_0(u)=\#_1(u)$ & $\#_0(v)=\#_1(v)$. Hence $\#_0(w) = \#_0(u)+\#_0(v) = \#_1(u)+\#_1(v) = \#_1(w)$

# Proving Correctness of Grammars

**Claim:** Let $L = \{ w \mid \#_0(w) = \#_1(w) \}$. Then, $L(G) = L$ where the productions of $G$ are: S → $0S1 \mid 1S0 \mid$ SS $\mid \varepsilon$

**Proof:** Proving $L(G) \supseteq L$ by induction on the length of strings.

Suppose $w \in L$. To show by induction on $|w|$ that $w \in L(G)$.
Base cases: $|w|=0$. $\varepsilon \in L(G)$. ✓ No string with $|w|=1$ in $L(G)$. ✓

Induction step: Let $n \geq 2$. Suppose $u \in L(G)$ for all $u \in L$ with $|u| < n$.
Let $w \in L$ be such that $|w|=n$; i.e., $\#_0(w)=\#_1(w)$.

Case $w=0u1$: Then $u \in L$ and $|u| < n$. By IH, $u \in L(G)$. i.e., $S \Rightarrow^* u$.

Hence, $S \Rightarrow 0S1 \Rightarrow^* 0u1 = w$. (Case $w=1u0$ is symmetric.)

Case $w=0u0$: Let $d_i = \#_0(i\text{-long prefix of } w) - \#_1(i\text{-long prefix of } w)$.
Then $d_1 = 1$, $d_n = 0$, $d_{n-1} = -1$. So $\exists\ 1 < m \leq n-1$ s.t., $d_m = 0$. i.e., $w=xy$, where $|x|, |y| < |w|$, and $x,y \in L$. By IH, $x,y \in L(G)$. Hence $S \Rightarrow SS \Rightarrow^* xy = w$.

(Case $w=1u1$ is symmetric.)

# Proving Correctness of Grammars

Often will need to strengthen the claim to include strings generated by every variable in the grammar

**Claim:** Let $L = \{ w \mid \#_0(w) = \#_1(w) \}$. Then, $L(G) = L$ where productions of $G$ are:

$$S \rightarrow AB \mid BA \mid \varepsilon$$
$$A \rightarrow 0 \mid AS \mid SA$$
$$B \rightarrow 1 \mid BS \mid SB$$

**Stronger Claim:**

A derives all strings $w$ s.t. $\#_0(w) = \#_1(w)+1$.

B derives all strings $w$ s.t. $\#_1(w) = \#_0(w)+1$.

S derives all strings $w$ s.t. $\#_0(w) = \#_1(w)$.

# Closure Properties for CFL

**Union:** If $L_1$ and $L_2$ are CFLs, so is $L_1 \cup L_2$.

Let $G_1 = (\Sigma, V_1, P_1, S_1)$, $G_2 = (\Sigma, V_2, P_2, S_2)$ with $V_1 \cap V_2 = \emptyset$.

Let $G = (\Sigma, V, P, S)$ with $V = V_1 \cup V_2 \cup \{S\}$, and

$P = P_1 \cup P_2 \cup \{ S \rightarrow S_1 \mid S_2 \}$. Then $L(G) = L(G_1) \cup L(G_2)$.

**Concatenation:** If $L_1$ and $L_2$ are CFLs, so is $L_1 L_2$.

Let $G_1 = (\Sigma, V_1, P_1, S_1)$, $G_2 = (\Sigma, V_2, P_2, S_2)$ with $V_1 \cap V_2 = \emptyset$.

Let $G = (\Sigma, V, P, S)$ with $V = V_1 \cup V_2 \cup \{S\}$, and

$P = P_1 \cup P_2 \cup \{ S \rightarrow S_1 S_2 \}$. Then $L(G) = L(G_1) L(G_2)$.

**Kleene Star:** If $L_1$ is a CFL, so is $L_1{}^*$.

Let $G_1 = (\Sigma, V_1, P_1, S_1)$.

Let $G = (\Sigma, V, P, S)$ with $V = V_1 \cup \{S\}$, and

$P = P_1 \cup \{ S \rightarrow \varepsilon \mid S S_1 \}$. Then $L(G) = L(G_1)^*$.

# Closure Properties for CFL

CFLs are **not** closed under intersection or complement

Intersection: $L_1 = \{\ 0^i 1^j 0^k \mid i{=}j\ \}$ & $L_1 = \{\ 0^i 1^j 0^k \mid j{=}k\ \}$ are CFLs. But it turns out that $L_1 \cap L_2 = \{\ 0^i 1^j 0^k \mid i{=}j{=}k\ \}$ is not a CFL!

Complement: If CFLs were to be closed under complementation, since they are already closed under union, they would have been closed under intersection!

# Grammars

Rewriting rules for generating strings from a "seed"

In an "unrestricted" grammar, the rules are of the form $\alpha \rightarrow \beta$ where $\alpha, \beta \in (\Sigma \cup V)^*$

Context-Free Grammar: Rewriting rules apply to individual variables (with no "context")