PART 2:

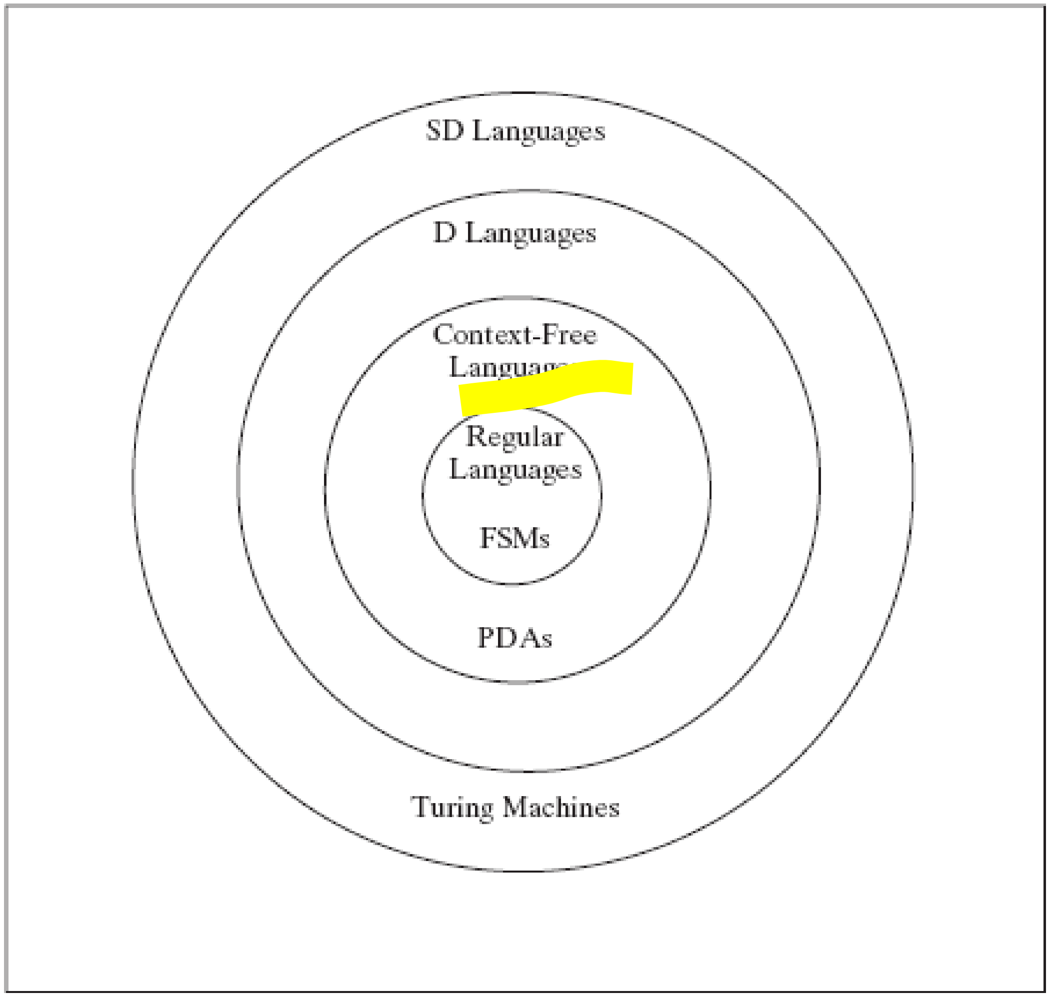
Automata:
PDA

Formal Language:
Context-Free Languages
Non-Context-Free Languages

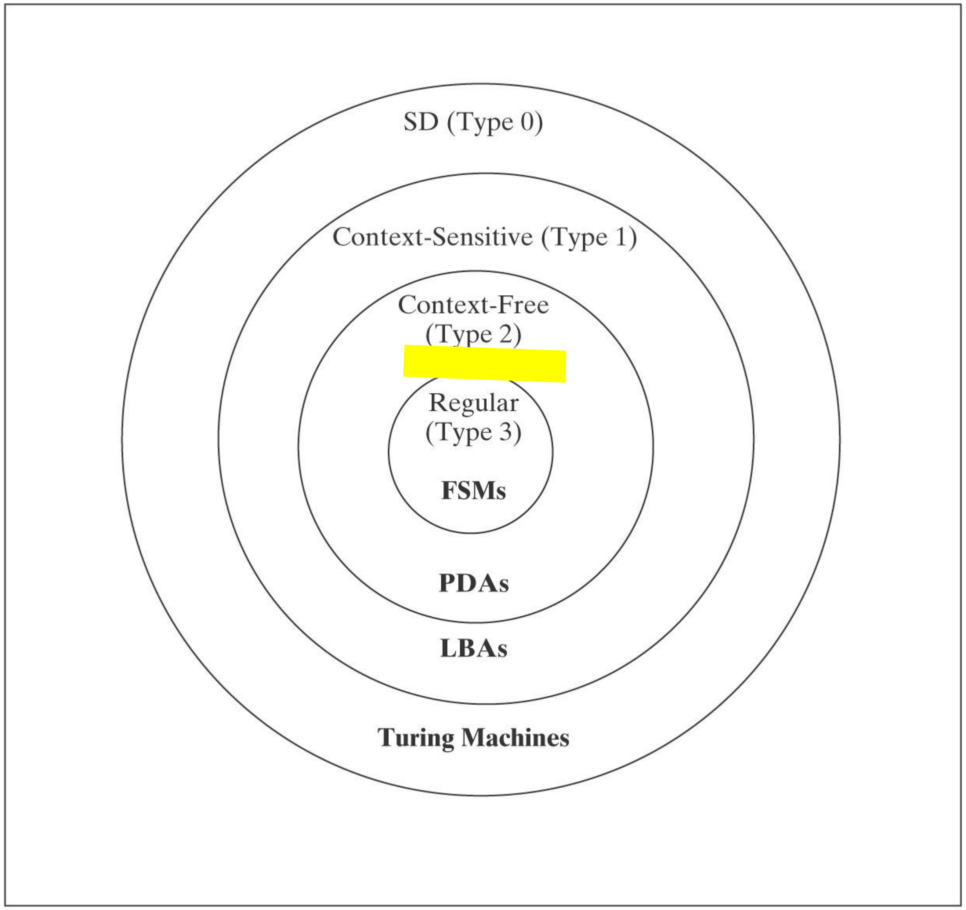
Grammar:
Context-Free Grammars

Context-Free Grammars

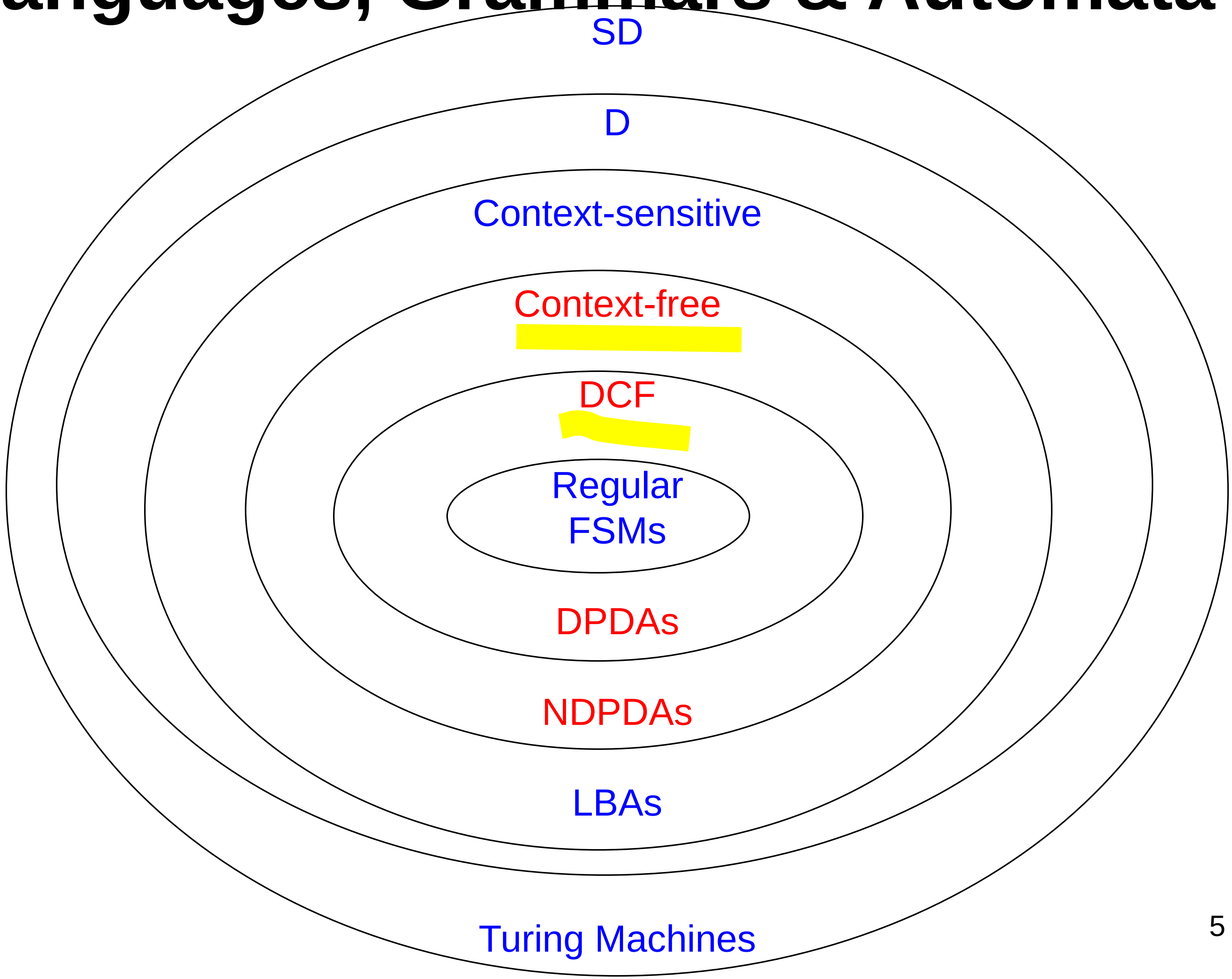
Languages, Grammars & Automata



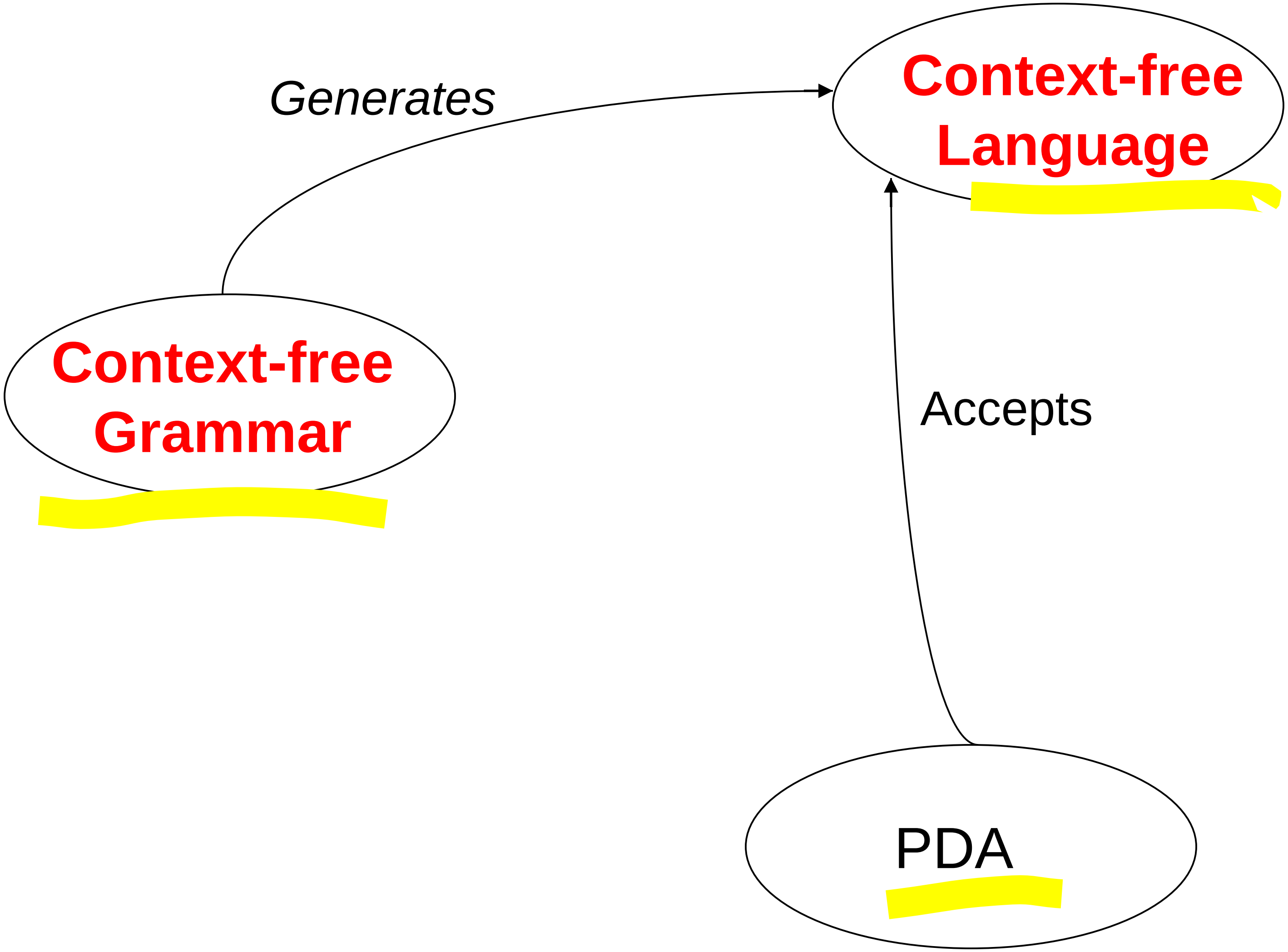
Languages, Grammars & Automata



Languages, Grammars & Automata



Context-free Grammars, Languages, and PDAs



Rewrite Systems

A rewrite system (or production system or rule-based system) is:

- a list of rules, and
- an algorithm for applying them.

Each rule has a left-hand side and a right hand side:

$$S \rightarrow \mathtt{a}S\mathtt{b}$$
$$\mathtt{a}S \rightarrow \varepsilon$$
$$\mathtt{a}S\mathtt{b} \rightarrow \mathtt{b}S\mathtt{ab}S\mathtt{a}$$

Grammars Generate Languages

A grammar is a rewrite system to derive/generate/define a language!

Grammars

A grammar is a set of **rules (productions)** that are stated in terms of two alphabets:

- A terminal alphabet, Σ, that contains the symbols that make up the strings in $L(G)$,
- A nonterminal alphabet, the elements of which will function as working symbols that will be used while the grammar is operating.
- A grammar has a unique start symbol, often called S.

Regular Grammars (RG)

In a **regular grammar**, all rules in *R* must:

1. have a left hand side that is a single nonterminal
2. have a right hand side that is:
   - ε, or
   - a single terminal, or
   - a single terminal followed by a single nonterminal.

Legal: $S \rightarrow \mathtt{a}$, $S \rightarrow \varepsilon$, and $T \rightarrow \mathtt{a}S$
Not legal: $S \rightarrow \mathtt{a}S\mathtt{a}$ and $\mathtt{a}S\mathtt{a} \rightarrow T$

Context-Free Grammars (CFG)

In a **context-free grammar**, all rules in *R* must:

1. have a left hand side that is a single nonterminal.
2. have a right hand side.

✓ No restrictions on the form of the right hand sides!

$$S \rightarrow \mathtt{ab}D\mathtt{e}FG\mathtt{ab}$$

Definition of Context-Free Grammars

A context-free grammar G is a quadruple (V, Σ, R, S) where:

- V is the rule alphabet, which contains nonterminals and terminals.
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite subset of $(V - \Sigma) \times V^*$,
  - ✓ All rules in R must have a left hand side that is a single nonterminal and have a right hand side.
- S (the start symbol) is an element of $V - \Sigma$.

Derivations Using A CFG

$x \Rightarrow_G y$ iff $x = \alpha A \beta$

and $A \rightarrow \gamma$ is in R

$y = \alpha\, \gamma\, \beta$

Sentential Forms

$w_0 \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \ldots \Rightarrow_G w_n$ is a *derivation* in G.

Let $\Rightarrow_G{}^*$ be the *reflexive, transitive closure* of $\Rightarrow_G$.

Example: CFG

$G = \{\{S, \mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, R, S\}$, where:

$R = \{$

$S \rightarrow \varepsilon$

$S \rightarrow \mathtt{c}$

$S \rightarrow \mathtt{a}S\mathtt{b}$

$\}$

ε

acb

aaabbb

aacbbb

Leftmost and Rightmost Derivations

- Left-most derivation:

  Always choose *left-most nonterminal* for expansion!

- Right-most derivation:

  Always choose *right-most nonterminal* for expansion!

Recursive Rules

- A rule is *recursive* iff it is $X \rightarrow w_1 Y w_2$, where $Y \Rightarrow_G{}^* w_3 X w_4$ for some w_1, w_2, w_3, and w in V^*.

- Recursive rules make a finite grammar to generate infinite set of strings!

Recursive Grammars

- A grammar is recursive iff it contains at least one recursive rule.

$$S \rightarrow (S)$$

$$S \rightarrow (T)$$
$$T \rightarrow (S)$$

Self-Embedding Rules

- A rule in a grammar G is *self-embedding* iff it is $X \rightarrow w_1 Y w_2$, where $Y \Rightarrow_G{}^* w_3 X w_4$ and both $w_1 w_3$ and $w_4 w_2$ are in Σ^+.

✓ *A nonempty string on each side of the nested X!*

✓ *Pairs of matching regions!* $uv^q xy^q z$

Self-Embedding Grammars

- A grammar is self-embedding iff it contains at least one self-embedding rule.

$S \rightarrow \mathtt{a}S\mathtt{a}$ is self-embedding

$S \rightarrow \mathtt{a}S$ is not self-embedding

$S \rightarrow \mathtt{a}T$ $\quad T \rightarrow S\mathtt{a}$ is self-embedding

Self-Embedding Grammars

- A self-embedding grammar G does not guarantees L(G) is regular.

- If a grammar G is not self-embedding then $L(G)$ is regular.

- If a language L has the property that every grammar that defines it is self-embedding, then L is not regular.

Example 11.1

Bal = { w $\in$ {), (}* : the parentheses are balanced}

CFG?

$S \rightarrow \varepsilon$
$S \rightarrow SS$
$S \rightarrow (S)$

( )
( ) ( )
(( ) ( ))

Example 11.2

$A^nB^n = \{\mathrm{a}^n\mathrm{b}^n : n \geq 0\}$

CFG?

$S \rightarrow \varepsilon$
$S \rightarrow \mathrm{a}S\mathrm{b}$

ε

ab

aaabbb

aabbb

The Language Generated by CFG

The language generated by CFG ***G***, denoted ***L*(*G*)**, is

the set of terminal strings that have derivations from the starting symbol.

$\{w \in \Sigma^* : S \Rightarrow_G^* w\}$.

Context-Free Languages (CFL)

A language *L* is ***context-free*** iff it is generated by some **context-free grammar** *G*.

CFL = CFG

Examples and Designing CFGs

Example 11.3

PalEven = $\{ww^R : w \in \{\texttt{a}, \texttt{b}\}^*\}$

CFG?

$G = \{\{S, \texttt{a}, \texttt{b}\}, \{\texttt{a}, \texttt{b}\}, R, S\}$, where:

$$R = \{\ S \rightarrow \texttt{a}S\texttt{a}$$
$$S \rightarrow \texttt{b}S\texttt{b}$$
$$S \rightarrow \varepsilon\ \}.$$

`ababbaba`

`ababba`

Example 11.4

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*: \#_{\mathtt{a}}(w) = \#_{\mathtt{b}}(w)\}$.

CFG?

$G = \{\{S, \mathtt{a}, \mathtt{b}\}, \{\mathtt{a}, \mathtt{b}\}, R, S\}$, where:

$$
\begin{aligned}
R = \{\ & S \rightarrow \mathtt{a}S\mathtt{b} \\
& S \rightarrow \mathtt{b}S\mathtt{a} \\
& S \rightarrow SS \\
& S \rightarrow \varepsilon\ \}.
\end{aligned}
$$

`ababbaba`

`ababba`

BNF (Backus Naur Form)

A notation for writing *practical* context-free grammars:

- The symbol | should be read as “or”.

  Example: $S \rightarrow \mathtt{a}S\mathtt{b} \mid \mathtt{b}S\mathtt{a} \mid SS \mid \varepsilon$

- Allow a nonterminal symbol to be any sequence of characters surrounded by angle brackets.

  Examples of nonterminals:

  <program>

  <variable>

Example 11.5

BNF for a Java Fragment

```
<block> ::= {<stmt-list>} | {}
<stmt-list> ::= <stmt> | <stmt-list> <stmt>
<stmt> ::= <block> | while (<cond>) <stmt> |
          if (<cond>) <stmt> |
          do <stmt> while (<cond>); |
          <assignment-stmt>; |
          return | return <expression> |
          <method-invocation>;
```

Designing Context-Free Grammars

- Generate related regions together:

  A^nB^n

- Generate concatenated regions:

  $A \rightarrow BC$

- Generate outside in:

  $A \rightarrow \mathrm{a}A\mathrm{b}$

Example 11.7

$L = \{\texttt{a}^n\texttt{b}^n\texttt{c}^m : n, m \geq 0\}$.

CFG?

$G = (\{S, N, C, \texttt{a}, \texttt{b}, \texttt{c}\}, \{\texttt{a}, \texttt{b}, \texttt{c}\}, R, S\}$ where:

$$
\begin{aligned}
R = \{\ & S \to NC \\
& N \to \texttt{a}N\texttt{b} \\
& N \to \varepsilon \\
& C \to \texttt{c}C \\
& C \to \varepsilon\ \}.
\end{aligned}
$$

ε
abc
aaabbbccc
aabbbcc

Example 11.8

L = { $a^{n_1}b^{n_1}a^{n_2}b^{n_2}\ldots a^{n_k}b^{n_k}$: $k \geq 0$ and $\forall i\ (n_i \geq 0)$}

ε
abab
aabbaaabbbabab

CFG?

G = ({S, M, a, b}, {a, b}, R, S} where:

R = { $S \rightarrow MS$
$S \rightarrow \varepsilon$
$M \rightarrow$ aMb
$M \rightarrow \varepsilon$}.

Parse Trees

- A tree representation for derivations!
- Equivalence of Parse Trees and Derivations!

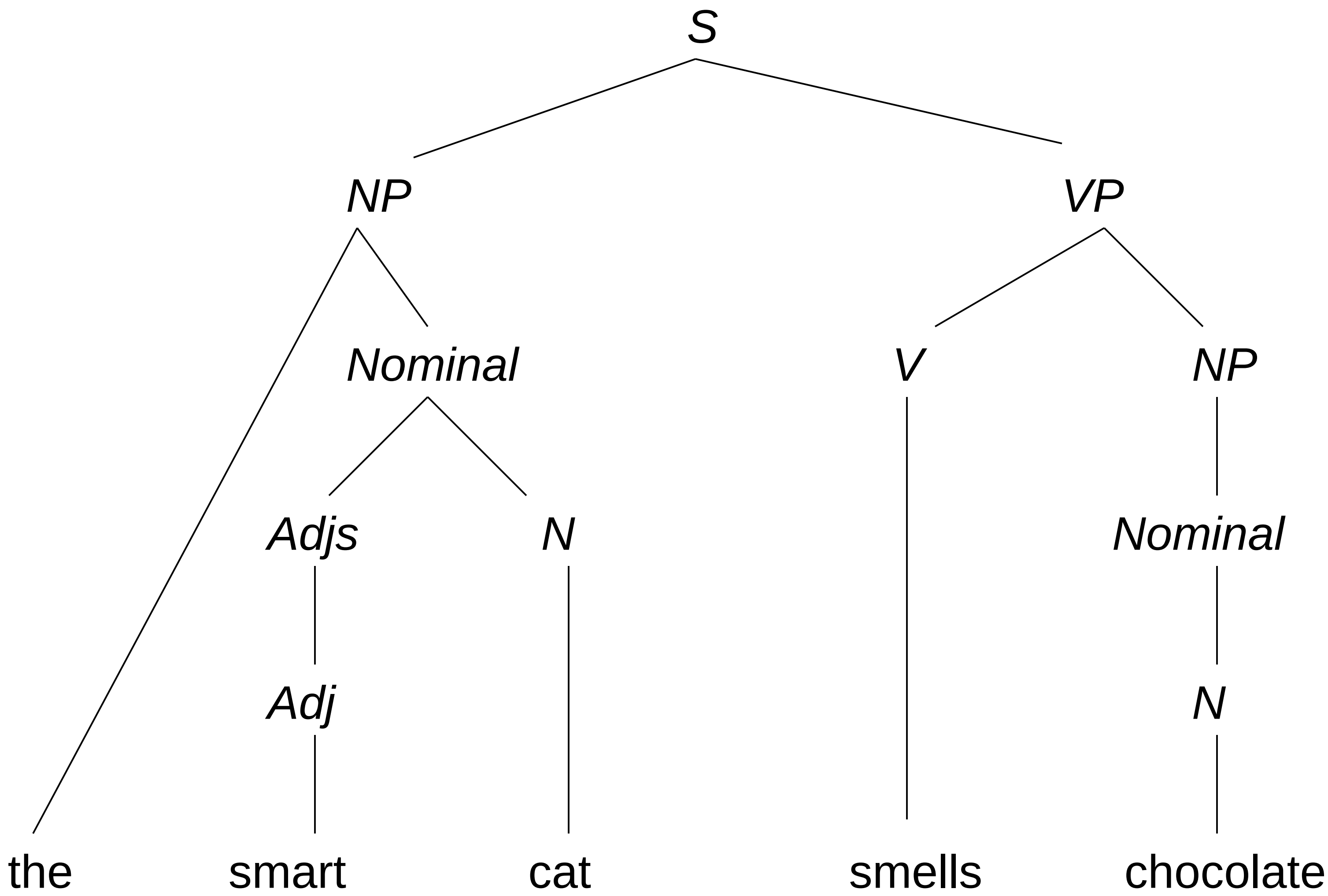
Parse Trees

A **parse tree**, *derived* by a grammar $G = (V, \Sigma, R, S)$, is a rooted, ordered tree in which:

- Every leaf node is labeled with an element of $\Sigma \cup \{\varepsilon\}$.
- The root node is labeled S.
- Every other node is labeled with some element of $V - \Sigma$.
- If m is a nonleaf node labeled X and the children of m are labeled $x_1, x_2, \ldots, x_n$, then R contains the rule $X \rightarrow x_1, x_2, \ldots, x_n$.

The yield of a parse tree is the string consisting of all leaves.

Example 11.11



Generative Capacity

Given a grammar *G*:

- *G*'s *weak generative capacity*, defined to be the set of strings, *L*(*G*), that *G* generates.
- *G*'s *strong generative capacity*, defined to be the set of parse trees that *G* generates.

Ambiguity & Inherent Ambiguity

Unambiguous Grammars

A grammar *G* is *unambiguous* iff **every string** derivable in *G* has **a single leftmost derivation**.

Ambiguous Grammars

A grammar G is *ambiguous* iff there is at least one string in *L*(*G*) for which *G* produces more than one parse tree.

- If G generates some string ambiguously.
- Two or more different leftmost (rightmost) derivations/ parse trees for some string.

✓ For most applications of context-free grammars, this is a problem!

Example 11.12

Bal = { w $\in$ {), (}* : the parentheses are balanced}

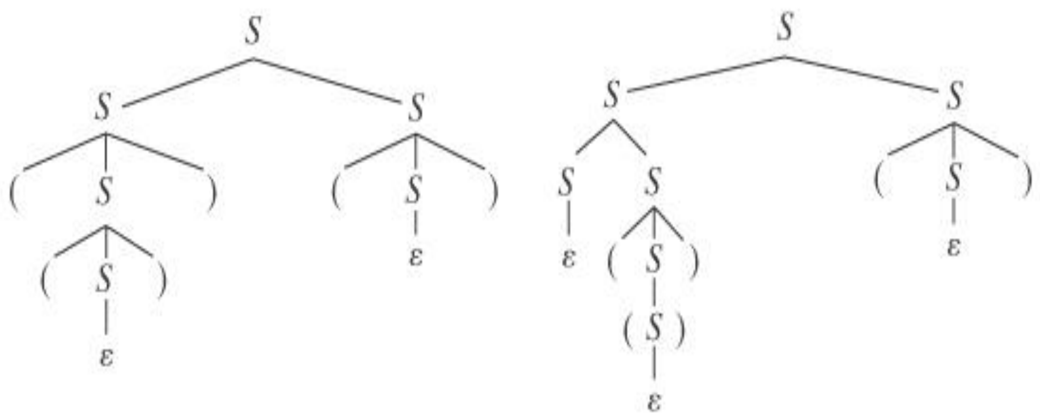
CFG:

$S \rightarrow \varepsilon$

$S \rightarrow SS$

$S \rightarrow (S)$

(( )) ( )



Ambiguous?

Example 11.13

L = { w $\in$ {a, b}* : w contains at least one a}

aaa

Regular expressions can be ambiguous too!

RE:

(a $\cup$ b)*a (a $\cup$ b)*

choose a from (a $\cup$ b)
choose a from (a $\cup$ b)
choose a

choose a
choose a from (a $\cup$ b)
choose a from (a $\cup$ b)

Example 11.13

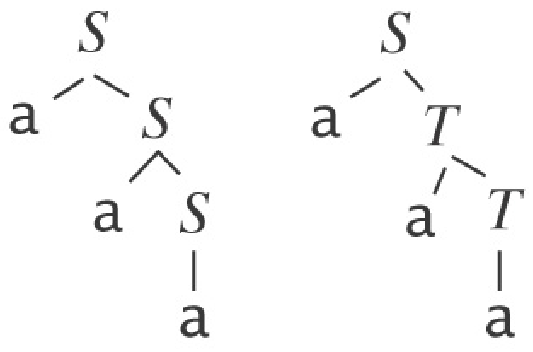
L = { w $\in$ {a, b}* : w contains at least one a}

aaa

Regular grammars can be ambiguous too!

RG:

$S \rightarrow$ a
$S \rightarrow$ bS
$S \rightarrow$ aS
$S \rightarrow$ aT
$T \rightarrow$ a
$T \rightarrow$ b
$T \rightarrow$ aT
$T \rightarrow$ bT



Example 11.14

An Ambiguous Grammar for Arithmetic Expressions:

CFG $G = (V, \Sigma, R, E)$, where

$V = \{+, *, (,), \mathtt{id}, E\}$,

$\Sigma = \{+, *, (,), \mathtt{id}\}$,

$R = \{$

$E \rightarrow E + E$

$E \rightarrow E * E$

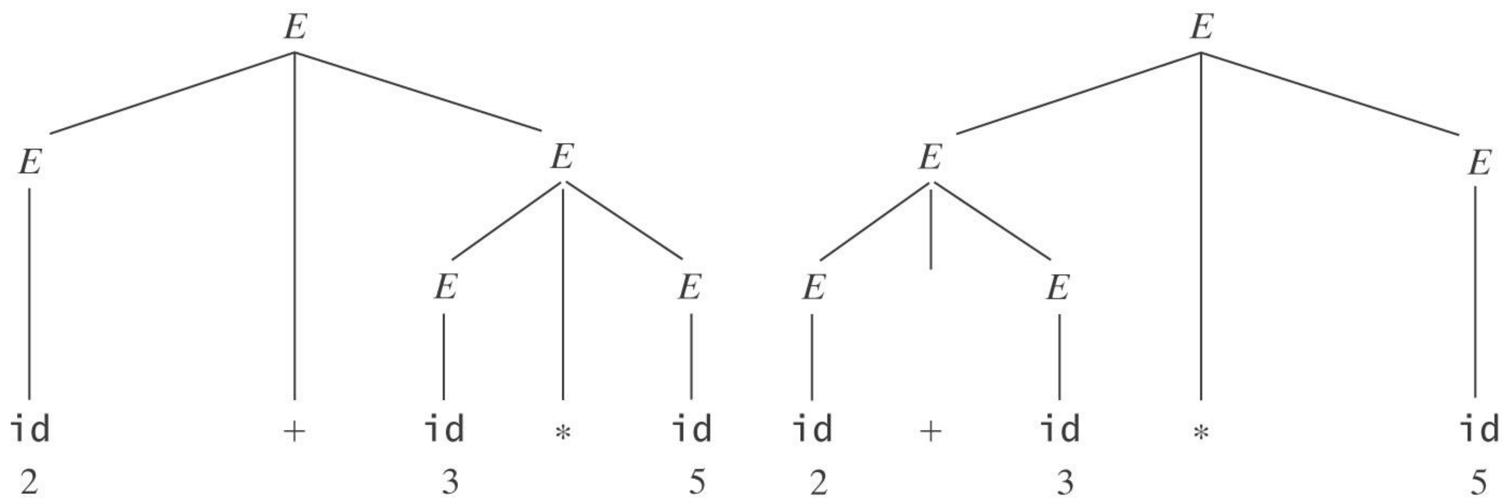
$E \rightarrow (E)$

$E \rightarrow \mathtt{id}\ \}$

Is G ambiguous?

2+ 3 * 5

Example 11.14



Inherent Ambiguous Languages

- Some languages have the property that every grammar for them is ambiguous.
- No unambiguous grammar exists!
- We call such languages ***inherently ambiguous*** *languages.*

Example11.15

$L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m: n, m \geq 0\} \cup \{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^m: n, m \geq 0\}$ is inherently ambiguous?

One grammar for L:

$S \rightarrow S_1 \mid S_2$
$S_1 \rightarrow S_1\mathtt{c} \mid A$ /* Generate all strings in $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^m\}$.
$A \rightarrow \mathtt{a}A\mathtt{b} \mid \varepsilon$
$S_2 \rightarrow \mathtt{a}S_2 \mid B$ /* Generate all strings in $\{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^m\}$.
$B \rightarrow \mathtt{b}B\mathtt{c} \mid \varepsilon$

Consider any string of the form $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n: n \geq 0\}$.
Two distinct derivations! L is inherently ambiguous!

Ambiguity & Inherent Ambiguity

Both of the following problems are undecidable:

- Given a context-free grammar *G*, is *G* ambiguous?

- Given a context-free language *L*, is *L* inherently ambiguous?

Reducing Ambiguity ?

Grammar structures lead to ambiguity:

- ε rules like $S \rightarrow \varepsilon$,

- Rules with symmetric right-hand sides, e.g.,

  $S \rightarrow SS$
  $E \rightarrow E + E$

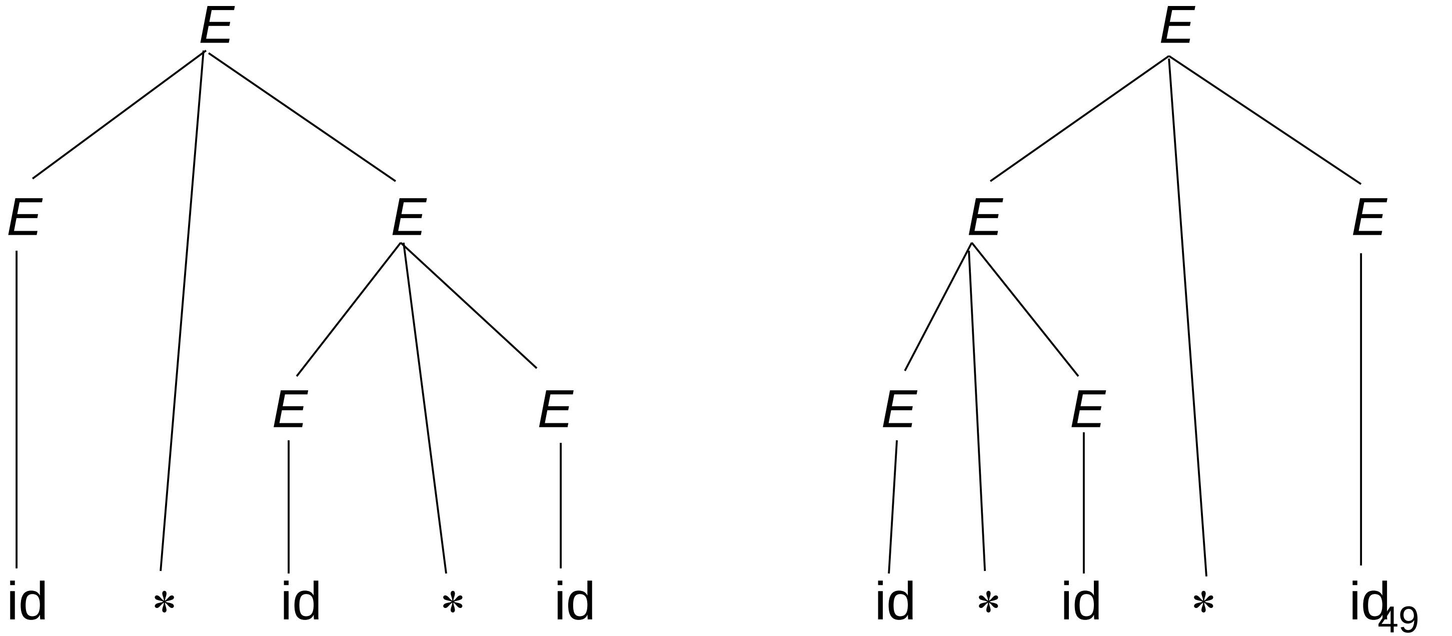
- Rule sets that lead to ambiguous attachment of optional postfixes.

Example 11.19

An Ambiguous Grammar for Arithmetic Expressions:

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow \texttt{id}$

Problem 1: Associativity

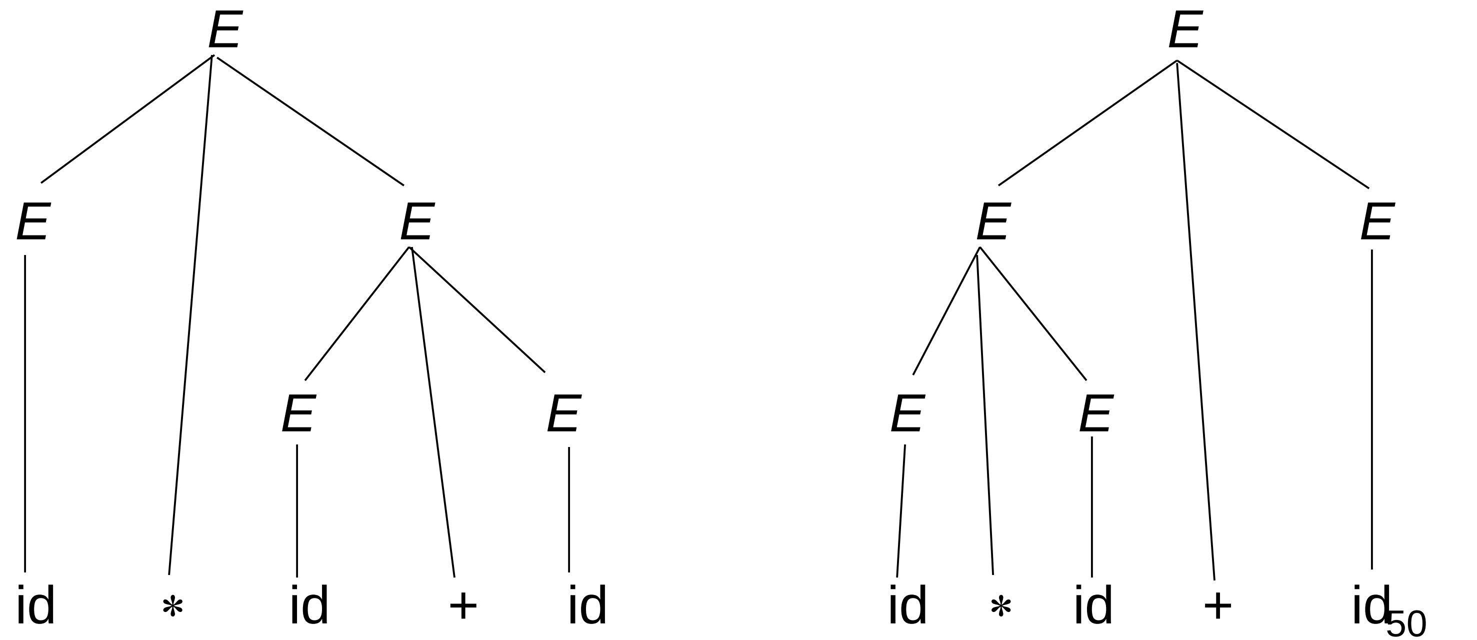


Example 11.19

An Ambiguous Grammar for Arithmetic Expressions:

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow \texttt{id}$

Problem 2: Precedence

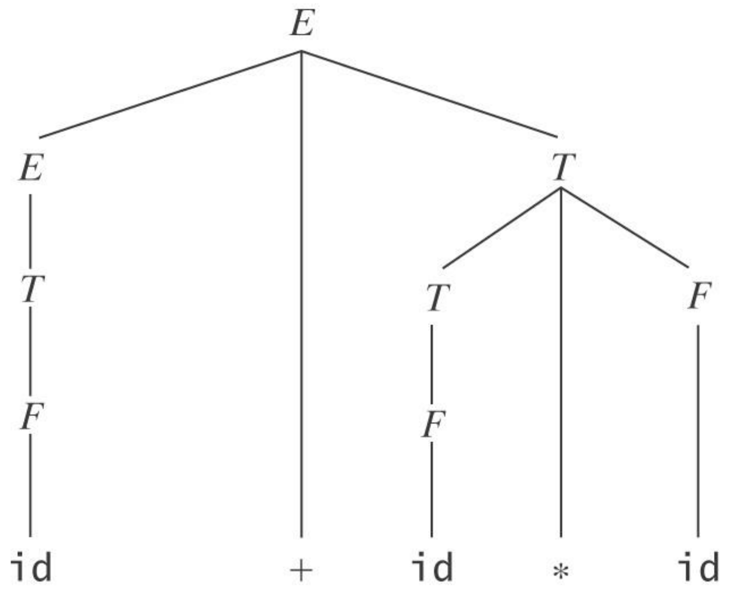


Example 11.19

An Unambiguous Grammar for Arithmetic Expressions:

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow id$

`id + id * id`



Ambiguous Attachment

The dangling else problem:

<stmt> ::= `if` <cond> `then` <stmt>
<stmt> ::= `if` <cond> `then` <stmt> `else` <stmt>

Ambiguous?

`if` cond_1 `then if` cond_2 `then` st_1 `else` st_2

Example 11.20

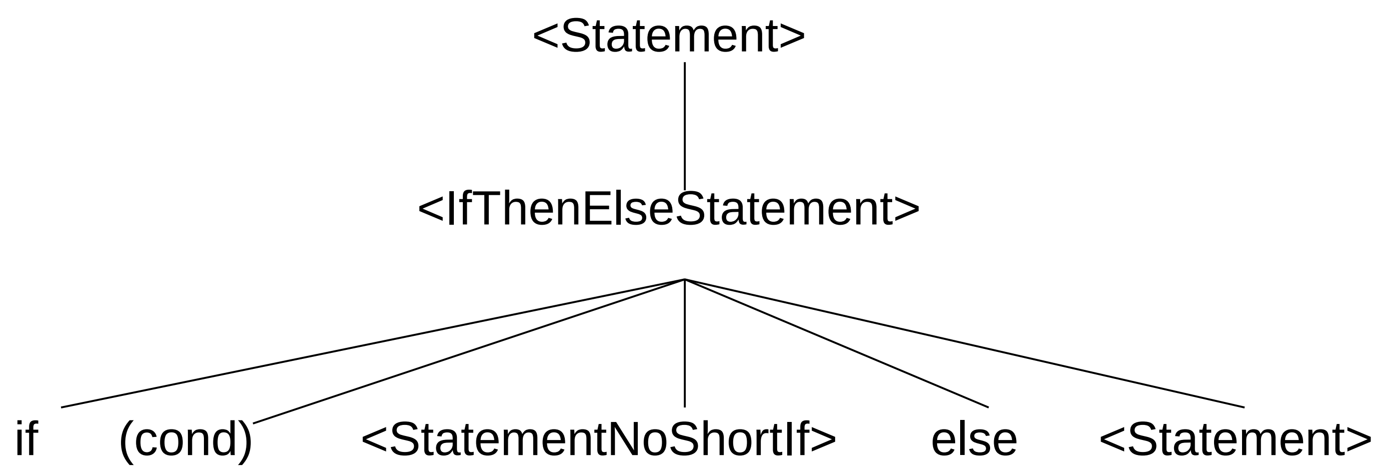
<Statement> ::= <IfThenStatement> | <IfThenElseStatement> |
<IfThenElseStatementNoShortIf>

<StatementNoShortIf> ::= <block> |
<IfThenElseStatementNoShortIf> | …

<IfThenStatement> ::= `if` (<Expression>) <Statement>

<IfThenElseStatement> ::= `if` (<Expression>)
<StatementNoShortIf> `else` <Statement>

<IfThenElseStatementNoShortIf> ::=
`if` (<Expression>) <StatementNoShortIf>
`else` <StatementNoShortIf>



Normal Forms for Grammars

Normal Forms for Grammars

Chomsky Normal Form, in which all rules are of one of the following two forms:

- $X \rightarrow a$, where $a \in \Sigma$, or
- $X \rightarrow BC$, where B and C are elements of $V - \Sigma$.

Advantages:

- Parsers can use binary trees.
- Exact length of derivations is known.

Normal Forms for Grammars

Greibach Normal Form, in which all rules are of the following form:

- $X \rightarrow a\ \beta$, where $a \in \Sigma$ and $\beta \in (V - \Sigma)^*$.

Advantages:

- Every derivation of a string *s* contains |*s*| rule applications.
- Greibach normal form grammars can easily be converted to pushdown automata with no ε-transitions. This is useful because such PDAs are guaranteed to halt.

Normal Forms Exist

Theorem 11.1 Given a CFG G, there exists an equivalent Chomsky normal form grammar G_C such that:

$$L(G_C) = L(G) - \{\varepsilon\}.$$

Proof Idea: Proof by construction.

Theorem 11.2 Given a CFG G, there exists an equivalent Greibach normal form grammar G_G such that:

$$L(G_G) = L(G) - \{\varepsilon\}.$$

Proof Idea: Proof by construction.

Normal Forms

$$E \rightarrow E + E$$
$$E \rightarrow (E)$$
$$E \rightarrow \mathtt{id}$$

Converting to Chomsky Normal Form:

$$E \rightarrow E\ E'$$
$$E' \rightarrow P\ E$$
$$E \rightarrow L\ E''$$
$$E'' \rightarrow E\ R$$
$$E \rightarrow \mathtt{id}$$
$$L \rightarrow ($$
$$R \rightarrow)$$
$$P \rightarrow +$$

Conversion doesn’t change weak generative capacity, but it may change strong generative capacity!

Comparing RL and CFL

Regular Languages	**Context-Free Languages**
regular expressions or regular grammars	context-free grammars
recognize	parse

vs.

Reading Assignment

Chapter 11:

Sections
11.1
11.2
11.3
11.6
11.7
11.8

In-Class Exercises

Chapter 11:

1 - b

2

3

6 – e & i

8

9

10