MA/CSSE 474
Theory of Computation

CFG Simplification, Correctness, Structure, and Ambiguity

What about Compilers Course?

- Is it like this one?
- What's it about?
- What will we do?
- What is the grade based on?

Recap: Context-Free Grammars

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

- V is the **rule alphabet** (**vocabulary**), 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 $(V - \Sigma) \times V^*$,
- S (the **start symbol**) is an element of $V - \Sigma$.

Rules are also called **productions**.

Example:

$(\{S, \mathtt{a}, \mathtt{b}\}, \ \{\mathtt{a}, \mathtt{b}\}, \ \{S \rightarrow \mathtt{a}S\mathtt{b}, \ S \rightarrow \varepsilon\}, \ S)$

Note: Some authors say that a CFG is (N, Σ, R, S), where N is the set of nonterminal symbols. In that case $V = N \cup \Sigma$. I may sometimes use N in this way. $N = V - \Sigma$.

Simplifying Context-Free Grammars

Remove non-productive and unreachable non-terminals.

Unproductive Nonterminals

removeunproductive(G: CFG) =

1. G' = G.
2. Mark every nonterminal symbol in G' as unproductive.
3. Mark every terminal symbol in G' as productive.
4. Until one entire pass has been made without any new symbol being marked do:
   For each rule $X \rightarrow \alpha$ in R do:
      If every symbol in α has been marked as productive and X has not yet been marked as productive then:
         Mark X as productive.
5. Remove from G' every unproductive symbol.
6. Remove from G' every rule that contains an unproductive symbol.
7. Return G'.

Unreachable Nonterminals

removeunreachable(G: CFG) =

1. G' = G.
2. Mark S as reachable.
3. Mark every other nonterminal symbol as unreachable.
4. Until one entire pass has been made without any new symbol being marked do:
   For each rule $X \rightarrow \alpha A\beta$ (where $A \in V - \Sigma$) in R do:
      If X has been marked as reachable and A has not then:
         Mark A as reachable.
5. Remove from G' every unreachable symbol.
6. Remove from G' every rule with an unreachable symbol on the left-hand side.
7. Return G'.

Proving the Correctness of a Grammar

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

$G = (\{S, a, b\}, \{a, b\}, R, S)$,

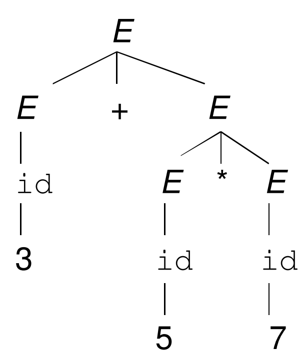
$R = \{ S \rightarrow a\, S\, b$
$\quad S \rightarrow \varepsilon$
$\}$

- Prove that *G* generates only strings in *L.*
- Prove that *G* generates all the strings in *L*.

Structure

Context free languages:

We care about structure.



Derivations

To capture structure, we must capture the path we took through the grammar. **Derivations** do that.

Example:

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

1 2 3 4 5 6

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()$

1 2 3 5 4 6

But the order of rule application doesn't matter.

Derivations

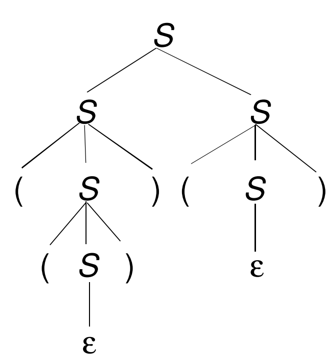
Parse trees capture essential structure:

1 2 3 4 5 6

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()$

1 2 3 5 4 6

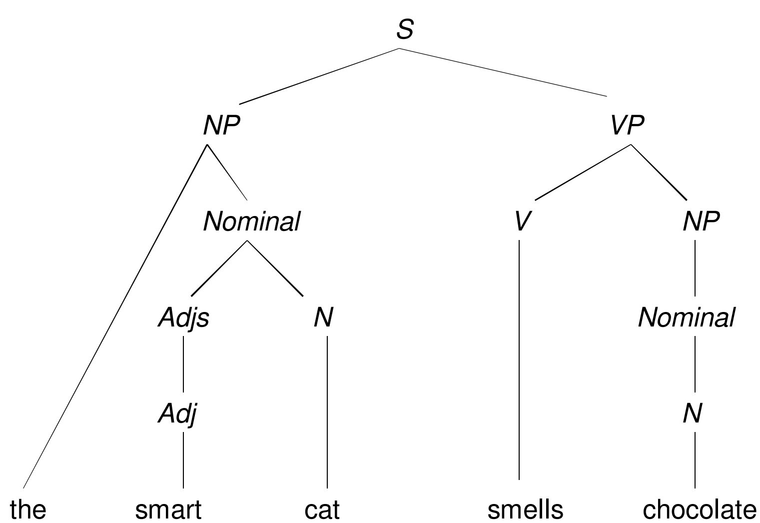


Parse Trees

A parse tree, derived from 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: $V - \Sigma$, and
- If m is a non-leaf node labeled X and the (ordered) children of m are labeled $x_1, x_2, \ldots, x_n$, then R contains the rule $X \rightarrow x_1\ x_2, \ldots\ x_n$.

Structure in English

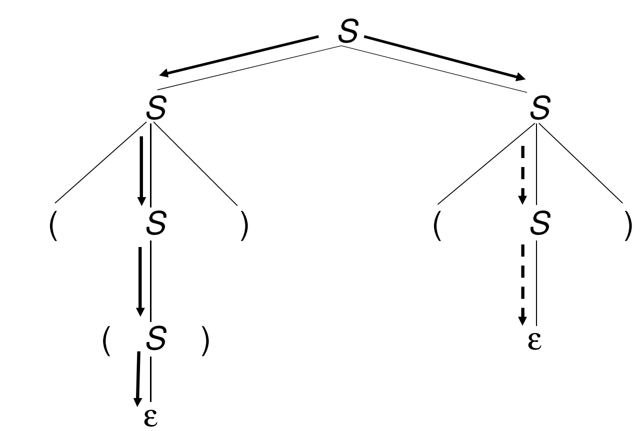


Generative Capacity

Because parse trees matter, it makes sense, given a grammar *G*, to distinguish between:

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

Algorithms Care How We Search



Algorithms for generation and recognition must be systematic. They typically use either the **leftmost derivation** or the **rightmost derivation.**

Derivations of The Smart Cat

- A left-most derivation is:

  S $\Rightarrow$ *NP VP* $\Rightarrow$ the *Nominal VP* $\Rightarrow$ the *Adjs N VP* $\Rightarrow$
  the *Adj N VP* $\Rightarrow$ the smart *N VP* $\Rightarrow$ the smart cat *VP* $\Rightarrow$
  the smart cat *V NP* $\Rightarrow$ the smart cat smells *NP* $\Rightarrow$
  the smart cat smells *Nominal* $\Rightarrow$ the smart cat smells *N* $\Rightarrow$
  the smart cat smells chocolate

- A right-most derivation is:

  S $\Rightarrow$ *NP VP* $\Rightarrow$ *NP V NP* $\Rightarrow$ *NP V Nominal* $\Rightarrow$ *NP V N* $\Rightarrow$
  NP V chocolate $\Rightarrow$ *NP* smells chocolate $\Rightarrow$
  the *Nominal* smells chocolate $\Rightarrow$
  the *Adjs N* smells chocolate $\Rightarrow$
  the *Adjs* cat smells chocolate $\Rightarrow$
  the *Adj* cat smells chocolate $\Rightarrow$
  the smart cat smells chocolate

Ambiguity

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

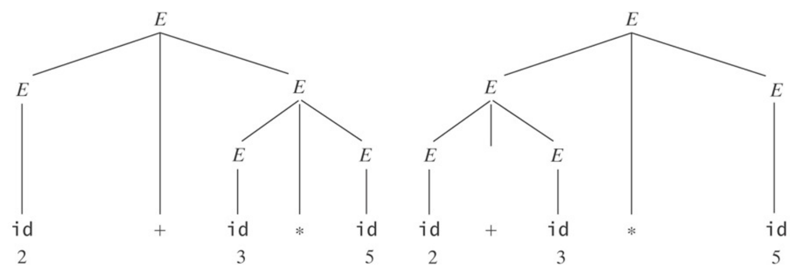
For many applications of context-free grammars, this is a problem.

Example: A programming language.

- If there can be two different structures for a string in the language, there can be two different meanings.
- Not good!

An Arithmetic Expression Grammar

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



Inherent Ambiguity

Some CF languages have the property that every grammar for them is ambiguous. We call such languages ***inherently ambiguous***.

Example:

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

Inherent Ambiguity

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

One grammar for L has the rules:

$S \rightarrow S_1 \mid S_2$

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

$S_2 \rightarrow aS_2 \mid B$ /* Generate all strings in $\{a^n b^m c^m\}$.
$B \rightarrow bBc \mid \varepsilon$

Consider any string of the form $a^n b^n c^n$.

It turns out that L is inherently ambiguous.

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?

But We Can Often Reduce Ambiguity

We can get rid of:

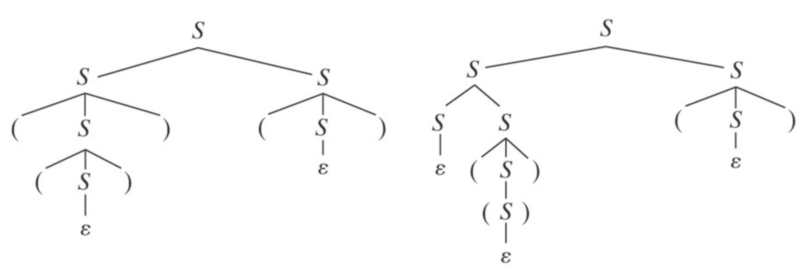
- some ε rules like $S \rightarrow \varepsilon$,
- rules with symmetric right-hand sides, e.g.,

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

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

A Highly Ambiguous Grammar

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



Resolving the Ambiguity with a Different Grammar

The biggest problem is the ε rule.

A different grammar for the language of balanced parentheses:

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

Nullable Nonterminals

Examples:

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

$S \rightarrow \mathrm{a}T\mathrm{a}$
$T \rightarrow A\ B$
$A \rightarrow \varepsilon$
$B \rightarrow \varepsilon$

A nonterminal X is ***nullable*** iff either:
(1) there is a rule $X \rightarrow \varepsilon$, or
(2) there is a rule $X \rightarrow PQR\ldots$ and P, Q, R, … are all nullable.

Nullable Nonterminals

A nonterminal X is ***nullable*** iff either:

(1) there is a rule $X \rightarrow \varepsilon$, or

(2) there is a rule $X \rightarrow PQR\ldots$ and P, Q, R, … are all nullable.

So compute N, the set of nullable nonterminals, as follows:

1. Set N to the set of nonterminals that satisfy (1).
2. Repeat until an entire pass is made without adding anything to N

   Evaluate all other nonterminals with respect to (2).

   If any nonterminal satisfies (2) and is not in N, insert it.

A General Technique for Getting Rid of ε-Rules

Definition: a rule is ***modifiable*** iff it is of the form:

$P \rightarrow \alpha Q \beta$, for some nullable Q.

removeEps(G: cfg) =

1. Let $G' = G$.
2. Find the set N of nullable nonterminals in G'.
3. Repeat until G' contains no modifiable rules that haven't been processed:

   Given the rule $P \rightarrow \alpha Q \beta$, where $Q \in N$,

   add the rule $P \rightarrow \alpha\beta$

   if it is not already present and if $\alpha\beta \neq \varepsilon$ and if $P \neq \alpha\beta$.
4. Delete from G' all rules of the form $X \rightarrow \varepsilon$.
5. Return G'.

$L(G') = L(G) - \{\varepsilon\}$

An Example

$G = \{\{S, T, A, B, C, \texttt{a}, \texttt{b}, \texttt{c}\}, \{\texttt{a}, \texttt{b}, \texttt{c}\}, R, S),$

$R = \{\ S \rightarrow \texttt{a}T\texttt{a}$

$T \rightarrow ABC$

$A \rightarrow \texttt{a}A \mid C$

$B \rightarrow B\texttt{b} \mid C$

$C \rightarrow \texttt{c} \mid \varepsilon\ \}$

removeEps(G: cfg) =
1. Let $G' = G$.
2. Find the set N of nullable nonterminals in G'.
3. Repeat until G' contains no modifiable rules that haven't been processed:
   Given the rule $P \rightarrow \alpha Q \beta$, where $Q \in N$,
   add the rule $P \rightarrow \alpha\beta$
   if it is not already present and if $\alpha\beta \neq \varepsilon$
   and if $P \neq \alpha\beta$.
4. Delete from G' all rules of the form $X \rightarrow \varepsilon$.
5. Return G'.

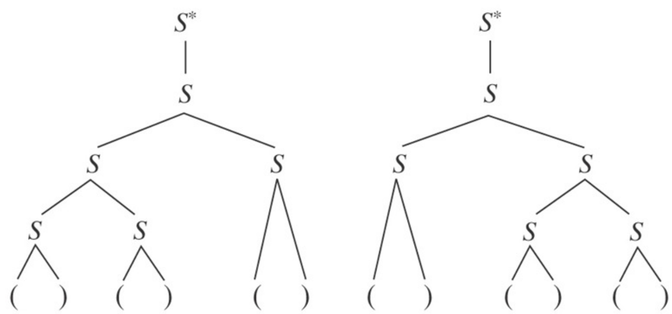
What If $\varepsilon \in L$?

atmostoneEps(G: cfg) =
1. G'' = *removeEps*(G).
2. If S_G is nullable then /* i. e., $\varepsilon \in L(G)$
   2.1 Create in G'' a new start symbol S^*.
   2.2 Add to $R_{G''}$ the two rules:
   $S^* \rightarrow \varepsilon$
   $S^* \rightarrow S_G$.
3. Return G''.

But There is Still Ambiguity

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

What about ()()() ?



Eliminating Symmetric Recursive Rules

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

Replace $S \rightarrow SS$ with one of:

$S \rightarrow SS_1$ /* force branching to the left
$S \rightarrow S_1S$ /* force branching to the right

So we get:

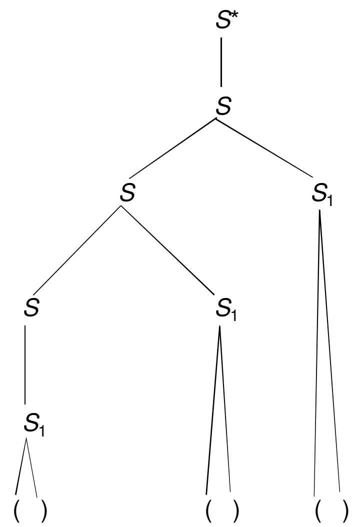
$S^* \rightarrow \varepsilon$
$S^* \rightarrow S$

$S \rightarrow SS_1$
$S \rightarrow S_1$
$S_1 \rightarrow (S)$
$S_1 \rightarrow ()$

Eliminating Symmetric Recursive Rules

So we get:

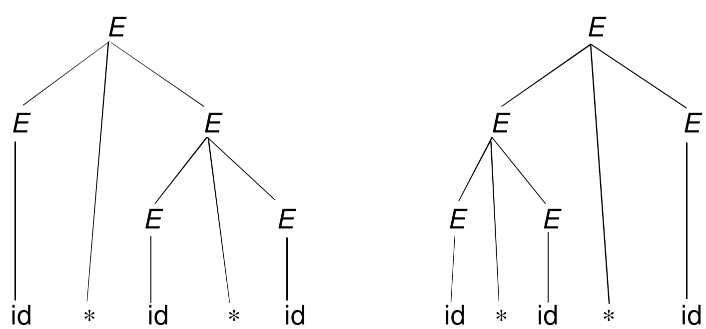
- $S^* \rightarrow \varepsilon$
- $S^* \rightarrow S$
- $S \rightarrow SS_1$
- $S \rightarrow S_1$
- $S_1 \rightarrow (S)$
- $S_1 \rightarrow ()$



Arithmetic Expressions

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow$ `id` }

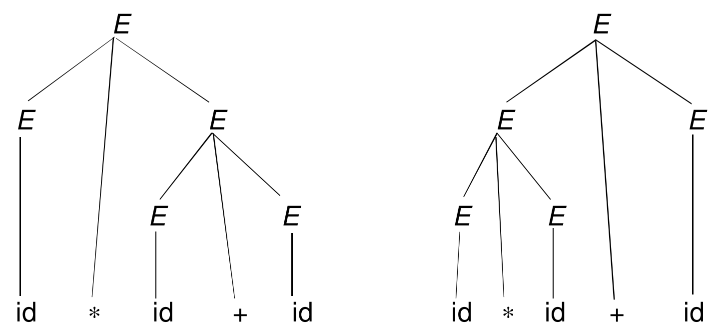
Problem 1: Associativity



Arithmetic Expressions

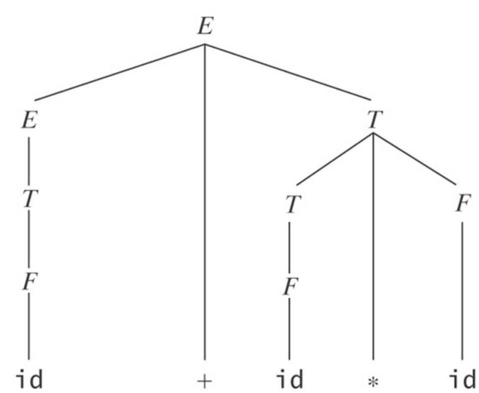
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow \texttt{id}$ }

Problem 2: Precedence



Arithmetic Expressions - A Better Way

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow \text{id}$



Ambiguous Attachment

The dangling else problem:

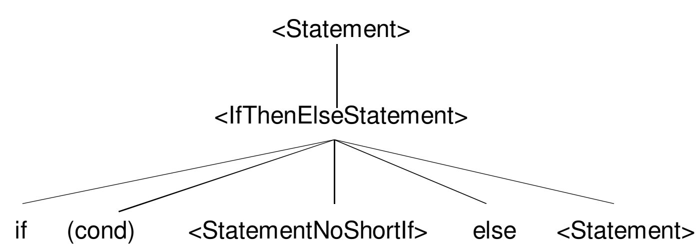
<stmt> ::= if <cond> then <stmt>
<stmt> ::= if <cond> then <stmt> else <stmt>

Consider:

if cond$_1$ then if cond$_2$ then st$_1$ else st$_2$

The Java Fix

<Statement> ::= <IfThenStatement> | <IfThenElseStatement> |
<IfThenElseStatementNoShortIf>
<StatementNoShortIf> ::= <block> |
<IfThenElseStatementNoShortIf> | …
<IfThenStatement> ::= if (<Expression>) <Statement>
<IfThenElseStatement> ::= if (<Expression>)
<StatementNoShortIf> else <Statement>
<IfThenElseStatementNoShortIf> ::=
if (<Expression>) <StatementNoShortIf>
else <StatementNoShortIf>



Going Too Far

S → *NP VP*
NP → the *Nominal* | *Nominal* | *ProperNoun* | *NP PP*
Nominal → *N* | *Adjs N*
N → `cat` | `girl` | `dogs` | `ball` | `chocolate` | `bat`
ProperNoun → `Chris` | `Fluffy`
Adjs → *Adj Adjs* | *Adj*
Adj → `young` | `older` | `smart`
VP → *V* | *V NP* | *VP PP*
V → `like` | `likes` | `thinks` | `hits`
PP → *Prep NP*
Prep → `with`

- `Chris likes the girl with the cat.`
- `Chris shot the bear with a rifle.`

Going Too Far

- `Chris likes the girl with the cat.`
- `Chris shot the bear with a rifle.`

- `Chris shot the bear with a rifle.`

Comparing Regular and Context-Free Languages

Regular Languages	Context-Free Languages
• regular exprs. or	
• regular grammars	• context-free grammars
• recognize	• parse