MA/CSSE 474
Theory of Computation

Remove Useless Nonterminals
Ambiguity
Normal forms

Your Questions?

- Previous class days' material
- Reading Assignments
- HW 9, 10 problems
- Anything else

This is quite a "complement" to Euclid!

Prove the Correctness of a Grammar

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

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

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

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

Simplify Context-Free Grammars

Remove non-productive and unreachable non-terminals.

Remove 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 nonterminal 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'.

Remove 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'.

Derivations and parse trees

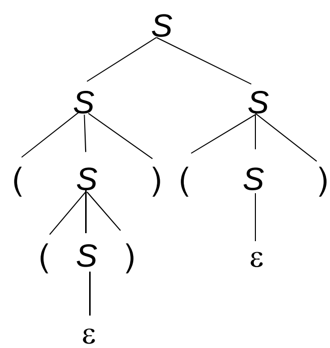
Parse trees capture essential structure:

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



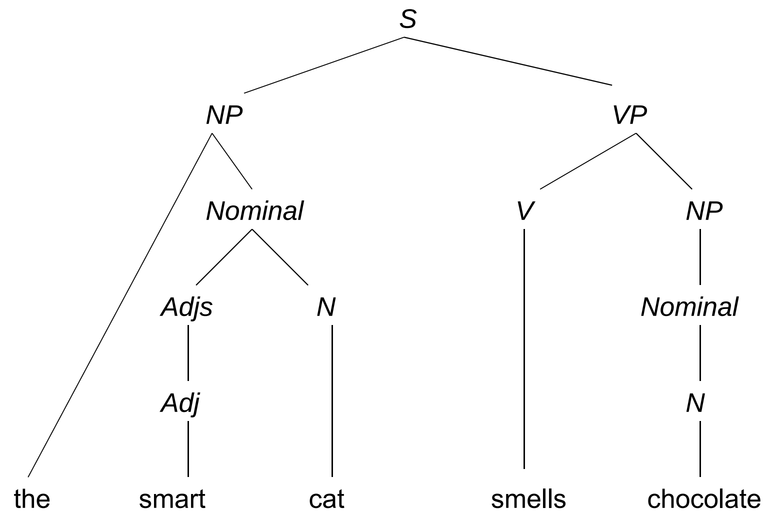
Parse Trees

A **parse tree**, (derivation 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 an element of $N = V - \Sigma$ 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

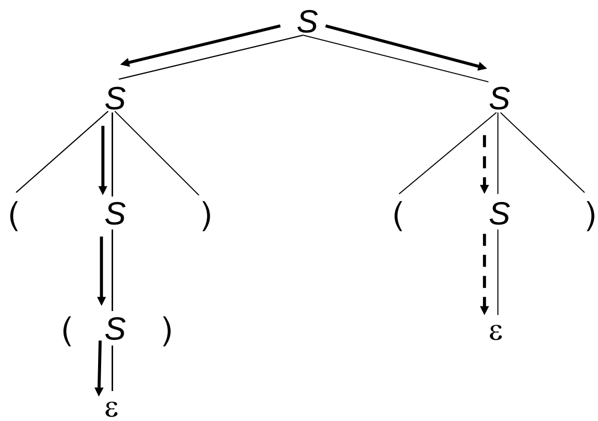


Generative Capacity

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 or Derive



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 ⇒ *NP VP* ⇒ the *Nominal VP* ⇒ the *Adjs N VP* ⇒
  the *Adj N VP* ⇒ the smart *N VP* ⇒ the smart cat *VP* ⇒
  the smart cat *V NP* ⇒ the smart cat smells *NP* ⇒
  the smart cat smells *Nominal* ⇒ the smart cat smells *N* ⇒
  the smart cat smells chocolate

- A right-most derivation is:
  S ⇒ *NP VP* ⇒ *NP V NP* ⇒ *NP V Nominal* ⇒ *NP V N* ⇒
  NP V chocolate ⇒ *NP* smells chocolate ⇒
  the *Nominal* smells chocolate ⇒
  the *Adjs N* smells chocolate ⇒
  the *Adjs* cat smells chocolate ⇒
  the *Adj* cat smells chocolate ⇒
  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*.

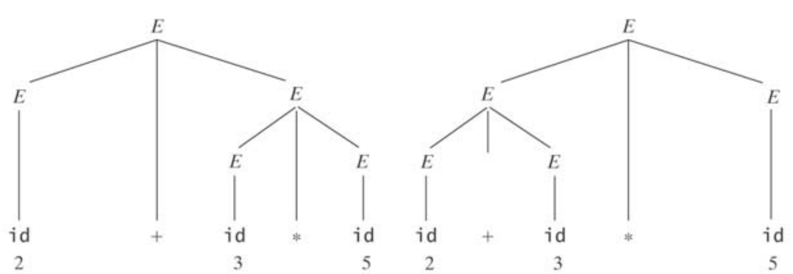
For many applications of context-free grammars, this is a problem.

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!

* Equivalently, more than one leftmost derivation, or more than one rightmost derivation.

An Arithmetic Expression Grammar

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



Inherent Ambiguity

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

Example:

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

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 these 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.

Ambiguity and undecidability

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?

Informal definition of *undecidable* for the first problem:
There is no algorithm (procedure that is guaranteed to always halt) that, given a grammar G, determines whether G is ambiguous.