Parse Trees

Read K & S 3.2
Read Supplementary Materials: Context-Free Languages and Pushdown Automata: Derivations and Parse Trees.
Do Homework 12.

Parse Trees

Regular languages:

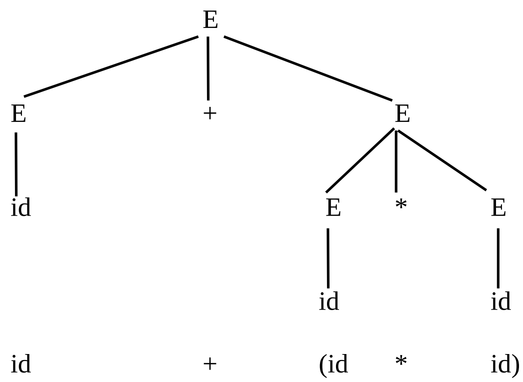
We care about recognizing patterns and taking appropriate actions.

Example: A parity checker

Structure

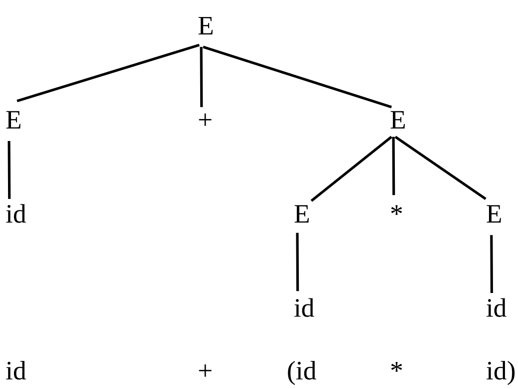
Context free languages:

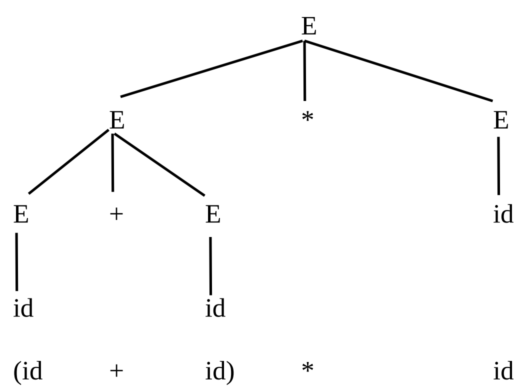
We care about structure.



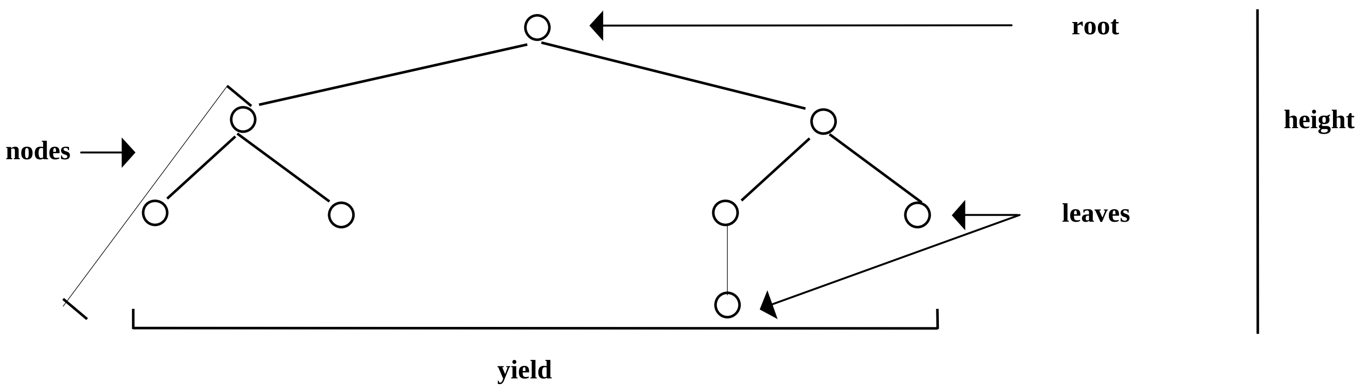
Parse Trees Capture Essential Structure

$E \rightarrow id$
$E \rightarrow E + E$
$E \rightarrow E * E$





Parse Trees are Just Trees



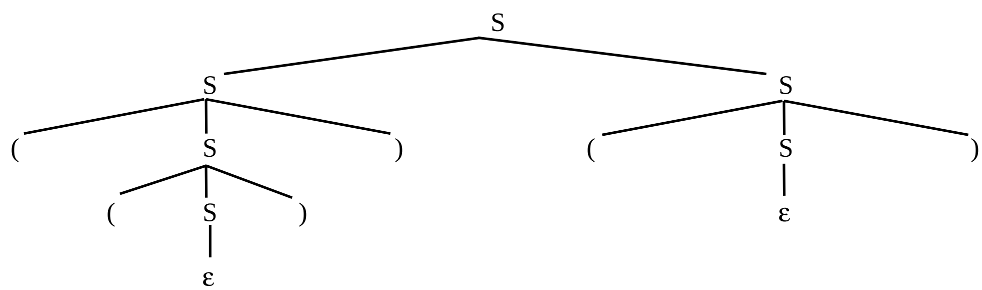
Leaves are all labeled with terminals or ε.
Other nodes are labeled with nonterminals.
A **path** is a sequence of nodes, starting at the root, ending at a leaf, and following branches in the tree.
The length of the yield of any tree T with height H and branching factor (fanout) B is ≤

Derivations

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

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

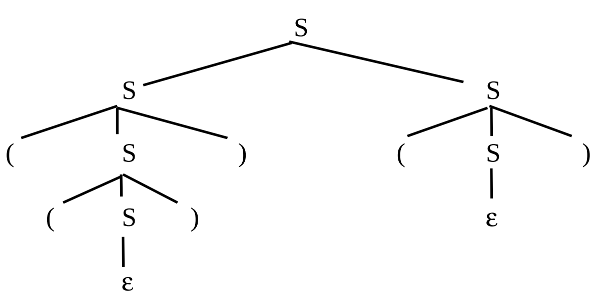
1 2 3 4 5 6
$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$
$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()$
1 2 3 5 4 6

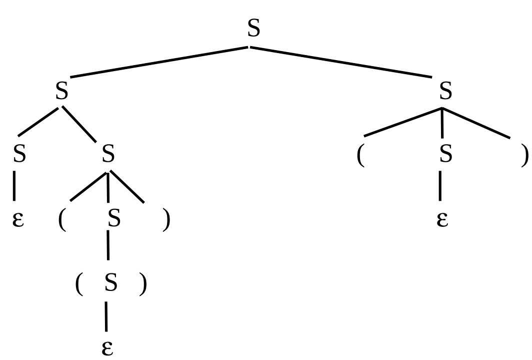


Alternative Derivations

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

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





Ordering Derivations

Consider two derivations:

1 2 3 4 5 6 7

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

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

1 2 3 4 5 6 7

We can write these, or any, derivation as

$D_1 = x_1 \Rightarrow x_2 \Rightarrow x_3 \Rightarrow \ldots \Rightarrow x_n$

$D_2 = x_1' \Rightarrow x_2' \Rightarrow x_3' \Rightarrow \ldots \Rightarrow x_n'$

We say that D_1 precedes D_2, written $D_1 < D_2$, if:

- D_1 and D_2 are the same length > 1, and
- There is some integer k, $1 < k < n$, such that:
  - for all $i \neq k$, $x_i = x_i'$
  - $x_{k-1} = x'_{k-1} = uAvBw$: $u, v, w \in V^*$, and $A, B \in V - \Sigma$
  - $x_k = uyvBw$, where $A \to y \in R$
  - $x_k' = uAvzw$ where $B \to z \in R$
  - $x_{k+1} = x'_{k+1} = uyvzw$

Comparing Several Derivations

Consider three derivations:

1 2 3 4 5 6 7

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

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

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

D1 < D2

D2 < D3

But D1 does not precede D3.

All three seem similar though. We can define similarity:

> D_1 is **similar** to D_2 iff the pair (D_1, D_2) is in the reflexive, symmetric, transitive closure of <.
>
> Note: similar is an equivalence class.

In other words, two derivations are similar if one can be transformed into another by a sequence of switchings in the order of rule applications.

Parse Trees Capture Similarity

1 2 3 4 5 6 7

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

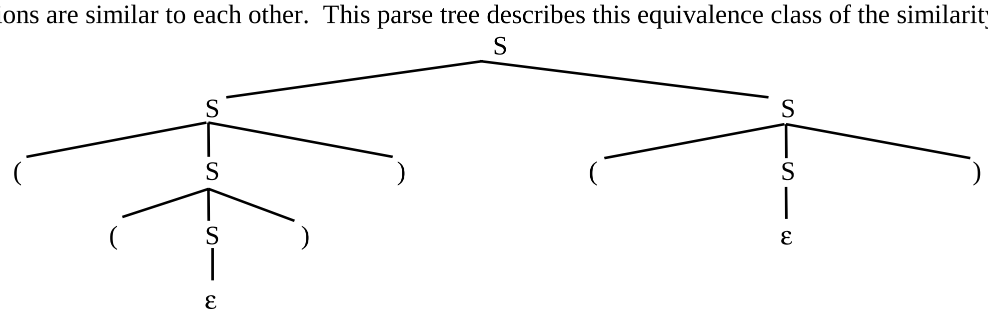
(2) $S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow ((S))(S) \Rightarrow (())(S) \Rightarrow (())()$

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

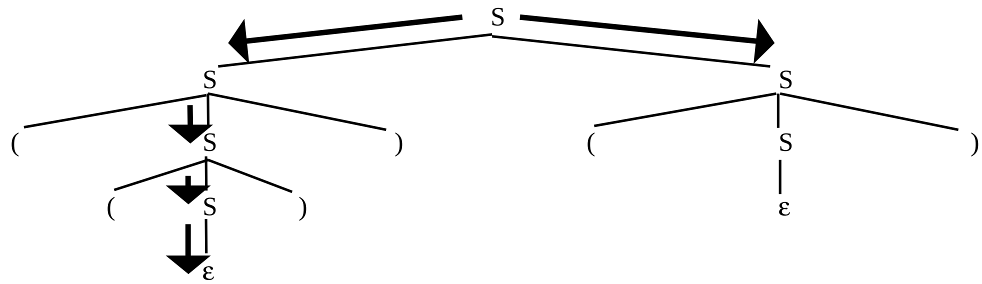
D1 < D2

D2 < D3

All three derivations are similar to each other. This parse tree describes this equivalence class of the similarity relation:



The Maximal Element of <



There's one derivation in this equivalence class that precedes all others in the class.

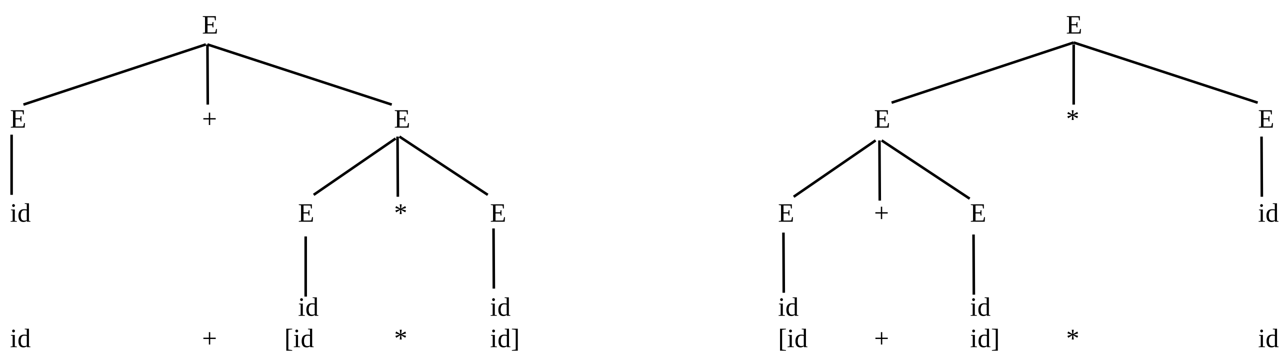
We call this the **leftmost derivation**. There is a corresponding rightmost derivation.

The leftmost (rightmost) derivation can be used to construct the parse tree and the parse tree can be used to construct the leftmost (rightmost) derivation.

Another Example

$E \rightarrow id$
$E \rightarrow E + E$
$E \rightarrow E * E$

(1) $E \Rightarrow E+E \Rightarrow E+E*E \Rightarrow E+E*id \Rightarrow E+id*id \Rightarrow id+id*id$
(2) $E \Rightarrow E*E \Rightarrow E*id \Rightarrow E+E*id \Rightarrow E+id*id \Rightarrow id+id*id$



Ambiguity

A grammar G for a language L is **ambiguous** if there exist strings in L for which G can generate more than one parse tree (note that we don't care about the number of derivations).

The following grammar for arithmetic expressions is ambiguous:

$E \rightarrow id$
$E \rightarrow E + E$
$E \rightarrow E * E$

Often, when this happens, we can find a different, unambiguous grammar to describe L.

Resolving Ambiguity in the Grammar

G = (V, Σ, R, E), where Parse : id + id * id

V = {+, *, (,), id, T, F, E},

Σ = {+, *, (,), id},

R = { $E \to E + T$

$E \to T$

$T \to T * F$

$T \to F$

$F \to (E)$

$F \to id$ }

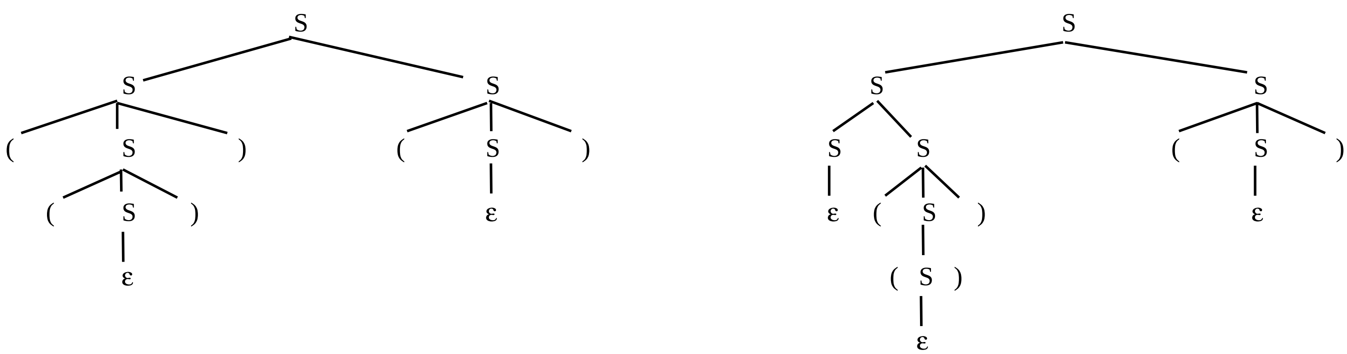
Another Example

The following grammar for the language of matched parentheses is ambiguous:

$S \to \varepsilon$

$S \to SS$

$S \to (S)$



Resolving the Ambiguity with a Different Grammar

One problem is the ε production.

A different grammar for the language of balanced parentheses:

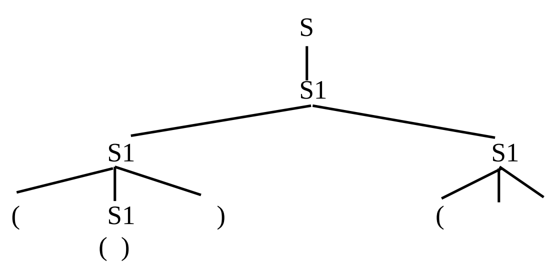
$S \to \varepsilon$

$S \to S_1$

$S_1 \to S_1\, S_1$

$S_1 \to (S_1)$

$S_1 \to ()$



A General Technique for Eliminating ε

If G is any context-free grammar for a language L and $\varepsilon \notin L$ then we can construct an alternative grammar G' for L by:

1. Find the set N of nullable variables:
   A variable V is **nullable** if either:
   there is a rule
   (1) $V \rightarrow \varepsilon$
   or there is a rule
   (2) $V \rightarrow PQR\ldots$ such that P, Q, R, … are all nullable
   So begin with N containing all the variables that satisfy (1). Evaluate all other variables with respect to (2). Continue until no new variables can be added to N.
2. For every rule of the form
   $P \rightarrow \alpha Q \beta$ for some Q in N, add a rule
   $P \rightarrow \alpha\beta$
3. Delete all rules of the form
   $V \rightarrow \varepsilon$

Sometimes Eliminating Ambiguity Isn't Possible

S → NP VP
NP → the NP1 | NP1 | NP2
NP1 → ADJ NP1 | N
NP2 → NP1 PP
ADJ → big | youngest | oldest
N → boy | boys | ball | bat | autograph
VP → V | V NP
VP → VP PP
V → hit| hits
PP → with NP

The boys hit the ball with the bat.

The boys hit the ball with the autograph.

Why It's Not Possible

- We could write an unambiguous grammar to describe L but it wouldn't always get the parses we want. Any grammar that is capable of getting all the parses will be ambiguous because the facts required to choose a derivation cannot be captured in the context-free framework.
  Example: Our simple English grammar
  [[The boys] [hit [the ball] [with [the bat]]]]
  [[The boys] [hit [the ball] [with [the autograph]]]]
- There is no grammar that describes L that is not ambiguous.
  Example: $L = \{a^n b^n c^m\} \cup \{a^n b^m c^m\}$

$S \rightarrow S_1 \mid S_2$
$S_1 \rightarrow S_1 c \mid A$
$A \rightarrow aAb \mid \varepsilon$
$S_2 \rightarrow aS_2 \mid B$
$B \rightarrow bBc \mid \varepsilon$

Now consider the strings $a^n b^n c^n$

They have two distinct derivations

Inherent Ambiguity of CFLs

A context free language with the property that all grammars that generate it are ambiguous is **inherently ambiguous**.

$L = \{a^n b^n c^m\} \cup \{a^n b^m c^m\}$ is inherently ambiguous.

Other languages that appear ambiguous given one grammar, turn out not to be inherently ambiguous because we can find an unambiguous grammar.

Examples: Arithmetic Expressions
Balanced Parentheses

Whenever we design practical languages, it is important that they not be inherently ambiguous.