Theory of Computing
SE-2112

Dr. Naushin Nower

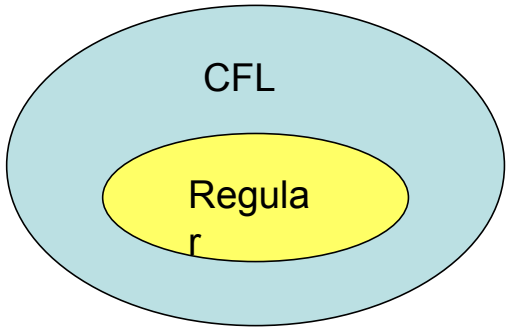
Context Free Grammars

A *context-free grammar* is a notation for describing languages.

It is more powerful than finite automata or RE's, but still cannot define all possible languages.

Every regular language is a CFL.

The class of regular languages is a proper subclass of CFLs.



Informal Example

Consider $L_{pal} = \{w \in \Sigma^* : w = w^R\}$

For example otto $\in L_{pal}$, madamimadam $\in L_{pal}$.

Let $\Sigma = \{0, 1\}$ and suppose L_{pal} were regular. Let n be given by the pumping lemma. Then $0^n10^n \in L_{pal}$. In reading 0^n the FA must make a loop. Omit the loop; contradiction.

Let's define L_{pal} inductively:

Basis: ε, 0, and 1 are palindromes.

Induction: If w is a palindrome, so are $0w0$ and $1w1$.

Informal Example

A context free grammar is a formal notation for expressing such (*Lpal*) recursive definitions of language. A grammar consists of one or more variables that represent classes of strings.

1. $P \rightarrow \varepsilon$ 2. $P \rightarrow 0$ 3. $P \rightarrow 1$

4. $P \rightarrow 0P0$

5. $P \rightarrow 1P1$

0 and 1 are *terminals*

P is a *variable* (or *nonterminal*, *category*) or *syntactic*

P is also the *start symbol* in this grammar.

1–5 are *productions* (or *rules*)

Context-Free Grammar (CFG)

Formally, A context-free grammar (CFG) G is a quadruple (V, T, P, S) where

V: a set of non-terminal symbols

T: a set of terminals ($V \cap \Sigma = Ø$)

P: a set of production ($P: V \rightarrow (V \cup T)^*$)

S: a start symbol

Example: $G_{pal} = (\{P\}, \{0, 1\}, A, P)$, where $A = \{P \rightarrow \varepsilon, P \rightarrow 0, P \rightarrow 1, P \rightarrow 0P0, P \rightarrow 1P1\}$.

Sometimes we group productions with the same head, e.g. $A = \{P \rightarrow \varepsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1\}$.

Example of CFG

Example: CFG for $\{ 0^n1^n \mid n \geq 1 \}$

Productions:

S -> 01

S -> 0S1

Basis: 01 is in the language.

Induction: if w is in the language, then so is 0w1.

Variables = *nonterminals* = a finite set of other symbols, each of which represents a language.

Terminals = symbols of the alphabet of the language being defined.

Start symbol = the variable whose language is the one being defined.

A *production* has the form variable -> string of variables and terminals.

Convention:

- A, B, C,… are variables.
- a, b, c,… are terminals.

Example of CFG

Here is a formal CFG for $\{ 0n1n \mid n \geq 1\}$.

Terminals = {0, 1}.

Variables = {S}.

Start symbol = S.

Productions =

S -> 01

S -> 0S1

G_{equ} = ({S}, {0, 1}, A, S), where A={S -> 01 , S -> 0S1}

Derivation Using Grammar

We apply the productions of a CFG to infer that certain strings are in the language of a certain variable.

There are two approaches to this inference.

- **Recursive inference**: is to use the rules from body to head
- **Derivations**, using productions from head to body

Recursive inference

We consider some inferences we can make using $tt1$

$$E \rightarrow I \mid E + E \mid E * E \mid (E)$$
$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

	String	Lang	Prod	String(s) used	a	I	5
(i)		-					
(ii)	b	I	6	-			
(iii)	b0	I	9	(ii)			
(iv)	b00	I	9	(iii)			
(v)	a	E	1	(i)			
(vi)	b00	E	1	(iv)			
(vii)	a + b00	E	2	(v), (vi)	(a + b00)	E	4
(viii)				(vii)			
(ix)	a *(a + b00)	E	3	(v), (viii)			

Derivations

Applying productions from head to body requires the definition of a new relational symbol: $\Rightarrow$

Let:

$G = (V, T, P, S)$ be a CFG

Let $\alpha A \beta$ *be a string of terminals and variables where* $A \in V$

$\alpha, \beta \subset (V \cup T)*$ and

$A \rightarrow \gamma$ *be a production of* P

Then we write

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

and say that $\alpha A \beta$ derives $\alpha \gamma \beta$.

Zero or more derivation steps

We define $\Rightarrow$ to be the reflexive and transitive closure of $\Rightarrow$ (*i.e.*, to denote zero or more derivation steps):

Basis: Let $\alpha \in (V \cup T)*$. Then $\alpha \Rightarrow \alpha$.

Induction: If $\alpha \overset{*}{\Rightarrow} \beta$ and $\beta \Rightarrow \gamma$, then $\alpha \overset{*}{\Rightarrow} \gamma$.

Example of Derivation

Derivation of $a*(a+b000)$ by G

$E \Rightarrow E*E \Rightarrow I*E \Rightarrow a*E \Rightarrow a*(E) \Rightarrow$

$a*(E+E) \Rightarrow a*(I+E) \Rightarrow a*(a+E) \Rightarrow a*(a+I) \Rightarrow$

$a*(a+I0) \Rightarrow a*(a+I00) \Rightarrow a*(a+b00)$

Note 1: At each step we might have several rules to choose from, e.g.

$I*E \Rightarrow a*E \Rightarrow a*(E)$, versus

$I*E \Rightarrow I*(E) \Rightarrow a*(E)$.

Note 2: Not all choices lead to successful derivations of a particular string, for instance

$E \Rightarrow E+E$ (at the first step)

won't lead to a derivation of $a*(a+b000)$.

Important: Recursive inference and derivation are equivalent. A string of terminals w is inferred to be in the language of some variable A iff $A \Rightarrow w$

Leftmost and Rightmost derivation

- In other to restrict the number of choices we have in deriving a string, it is often useful to require that at each step we replace the leftmost (or rightmost) variable by one of its production rules
- Leftmost derivation $\Rightarrow_{lm}$: Always replace the left-most variable by one of its rule-bodies
- Rightmost derivation $\Rightarrow_{rm}$: Always replace the rightmost variable by one of its rule-bodies.

EXAMPLES

1− Leftmost derivation: previous example

2− Rightmost derivation:

$E \Rightarrow_{rm} E*E$

$E*(E+E) \Rightarrow_{rm} E*(E+I)$

$E*(E+I00) \Rightarrow_{rm} E*(E+b00)$

$E*(a+b00) \Rightarrow_{rm} I*(a+b00)$

$E*(E) \Rightarrow_{rm} E*(E+I0)$

$E*(I+b00) \Rightarrow_{rm} a*(a+b00)$

We can conclude that $E \Rightarrow^{*}_{rm} a*(a+b00)$

The Language of the Grammar

If $G(V, T, P, S)$ is a CFG, then the language of G is

$$L(G) = \{w \text{ in } T* \mid S \stackrel{*}{\Rightarrow}_G w\}$$

i.e., the set of strings over G derivable from the start symbol.

If L is a CFG, we call $L(G)$ a context-free language.

Example: $L(Gpal)$ is a context-free language.

Theorem

A string $w \in \{0, 1\}*$ **is in** $L(Gpal)$ **iff** $w = wR$.

Proof : ($\supseteq$-direction.) Suppose $w = wR$, *i.e.*, that w is a palindrome. We show by induction on $|w|$ that $w \in L(Gpal)$

Basis: Basis: $|w| = 0$, or $|w| = 1$. Then w is ε, 0, or 1. Since $P \to \varepsilon, P \to 0$, and $P \to 1$ are productions, we conclude that $P \Rightarrow w$ in all base cases.

Induction: Suppose $|w| \geq 2$. Since $w = wR$, we have $w = 0x0$, or $w = 1x1$, and $x = xR$.

If $w = 0x0$ we know from the IH that $P \Rightarrow x$. Then

$$P \Rightarrow 0P0 \overset{*}{\Rightarrow} 0x0 = w$$

Thus $w \in L(Gpal)$. The case for w = 1x1 is similar.

Proof

($\subseteq$-direction.) We assume $w \in L(Gpal)$ and we prove that $w = wR$.

Since $w \in L(Gpal)$, we have $P \Rightarrow^* w$. We do an induction of the length of $\Rightarrow$.

Basis: The derivation $P \Rightarrow^* w$ is done in one step. Then w must be $\varepsilon, 0$, or 1, all palindromes.

Induction: Let $n \geq 1$, and suppose the derivation takes $n + 1$ steps. Then we must have

$w = 0x0 \Leftarrow^* 0P0 \Leftarrow P$

or

$w = 1x1 \Leftarrow^* 1P1 \Leftarrow P$

where the second derivation is done in n steps. By the IH x is a palindrome, and the inductive proof is complete.

Sentential Forms

Derivation from the start symbol produce strings that have a special role. We call these sentential forms.

Let G= (V, T, P, S) be a CFG, and $\alpha \in (V \cup T)*$. If

$$S \overset{*}{\Rightarrow} \alpha$$

we say α is a sentential form.

If $S \Rightarrow lm\, \alpha$ we say that α is a **left-sentential form**, and if $S \Rightarrow rm\, \alpha$ we say that is a right-sentential form.

Note: $L(G)$ is those sentential forms that are in $T*$.

Example

Recall $G1: E$

1− Then $E * (I + E)$ is a sentential form since

$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E) \Rightarrow E * (I + E)$$

This derivation is neither leftmost, nor right-most.

2− $a * E$ left-sentential form, since

$$E \Rightarrow E * E \Rightarrow I * E \Rightarrow a * E$$

3− $E * (E + E)$ is a right-sentential form since

$$E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E)$$

Parse Tree

Definition: Let G = (V, T, P, S) be a CFG. A tree is a derivation (or parse) tree if:

- Every interior node is labeled by a variable in V
- Each leaf is labeled by either a variable, a terminal or ε. However if leaf is ε, then it must be the only child of its parent.
- The label of the root is S
- If a interior node is labeled A and its children are labeled X1, X2,…, Xn, from left to right, then

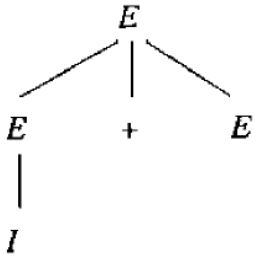
  A –> X1, X2,…, Xn

  must be a production in P
- If a vertex has label ε, then that vertex is a leaf and the only child of its' parent

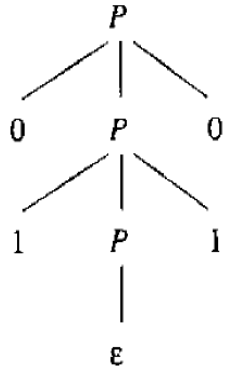
More Generally, a derivation tree can be defined with any non-terminal as the root.

Parse Tree..

A parse tree showing the derivation of I+E from E



A parse tree showing the derivation of P ⇒ 0110



Parse Trees

Parse trees are trees labeled by symbols of a particular CFG.

Leaves: labeled by a terminal or ε.

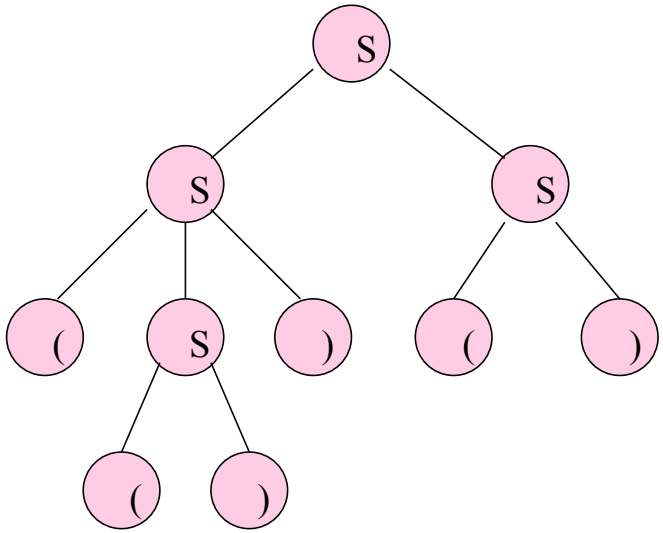
Interior nodes: labeled by a variable.

- Children are labeled by the right side of a production for the parent.

Root: must be labeled by the start symbol.

Example: Parse Tree

S -> SS | (S) | ()



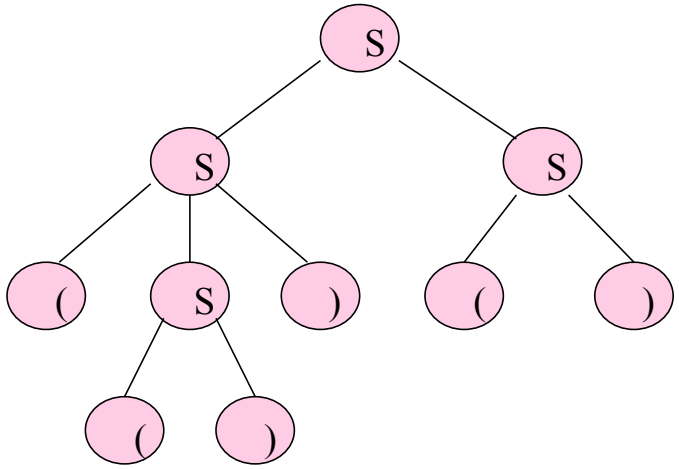
Yield of a Parse Tree

The concatenation of the labels of the leaves in left-to-right order

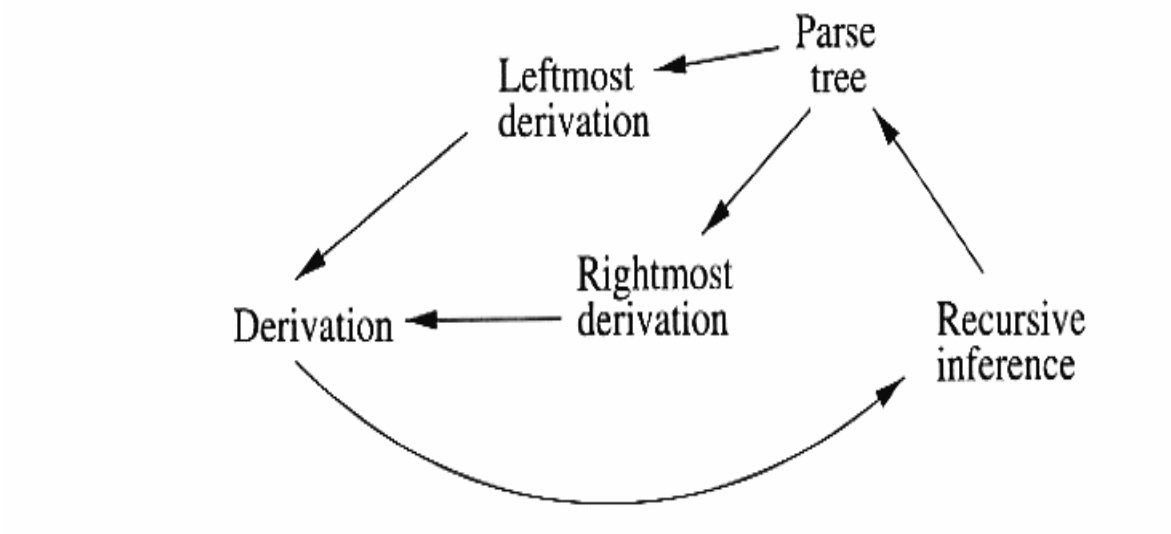
- That is, in the order of a preorder traversal.

is called the *yield* of the parse tree.

Example: yield of is (())()



Inference Derivation and Parse Tree



From Inference to Trees

We'll prove:

1. If A =>*lm w, then there is a parse tree with root A and yield w.
2. If there is a parse tree with root labeled A and yield w, then A =>*lm w.

Theorem 5.12: Let $G = (V, T, P, S)$ be a CFG. If the recursive inference procedure tells us that terminal string w is in the language of variable A, then there is a parse tree with root A and yield w.

Theorem 5.14: Let $G = (V, T, P, S)$ be a CFG, and suppose there is a parse tree with root labeled by variable A and with yield w, where w is in T^*. Then there is a leftmost derivation $A \underset{lm}{\overset{*}{\Rightarrow}} w$ in grammar G.

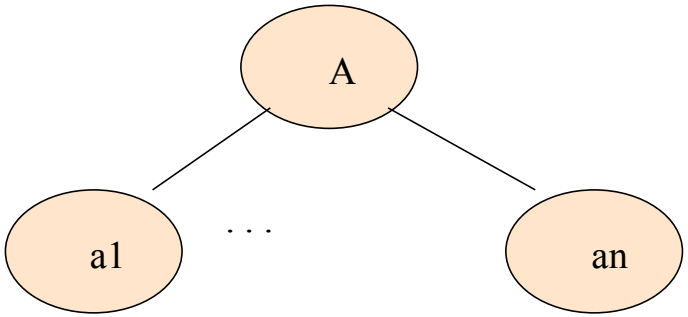
Proof: Part 1

Given a leftmost derivation of a terminal string, we need to prove the existence of a parse tree.

The proof is an induction on the length of the derivation.

Part 1 – Basis

If A =>*lm a1…an by a one-step derivation, then there must be a parse tree



Part 1 – Induction

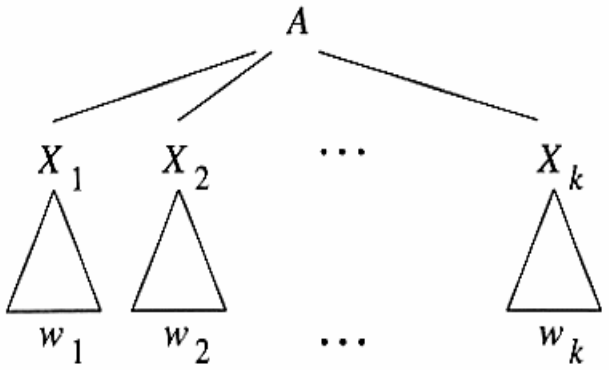
INDUCTION: Suppose that the fact w is in the language of A is inferred after $n+1$ inference steps, and that the statement of the theorem holds for all strings x and variables B such that the membership of x in the language of B was inferred using n or fewer inference steps. Consider the last step of the inference that w is in the language of A. This inference uses some production for A, say $A \to X_1 X_2 \cdots X_k$, where each X_i is either a variable or a terminal.

We can break w up as $w_1 w_2 \cdots w_k$, where:

1. If X_i is a terminal, then $w_i = X_i$; i.e., w_i consists of only this one terminal from the production.

2. If X_i is a variable, then w_i is a string that was previously inferred to be in the language of X_i. That is, this inference about w_i took at most n of the $n+1$ steps of the inference that w is in the language of A. It cannot take all $n+1$ steps, because the final step, using production $A \to X_1 X_2 \cdots X_k$, is surely not part of the inference about w_i. Consequently, we may apply the inductive hypothesis to w_i and X_i, and conclude that there is a parse tree with yield w_i and root X_i.

Induction – (2)



Proof – Part 2

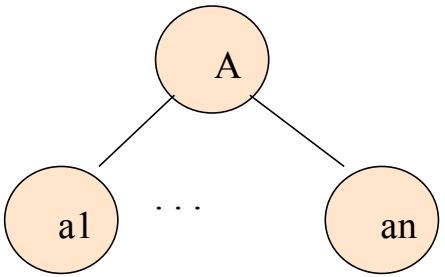
We perform induction on the height of the tree.

Induction on the *height* (length of the longest path from the root) of the tree.

Basis: height 1. Tree looks like

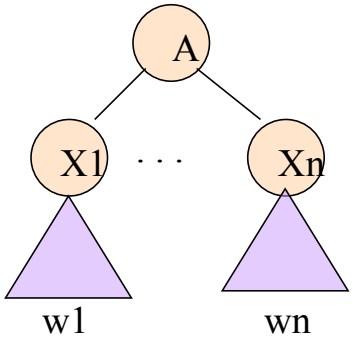
A -> a1…an must be a production.

Thus, A =>*lm a1…an.



Part 2 – Induction

If the height of the tree is n and n>1, then it looks like



By IH, Xi =>*lm wi.

- Note: if Xi is a terminal, then Xi = wi.

Thus, A =>lm X1…Xn =>*lm w1X2…Xn =>*lm
w1w2X3…Xn =>*lm … =>*lm w1…wn.

Applications of Context-free Grammars

Grammars are used to describe programming languages.

Example 5.19: Typical languages use parentheses and/or brackets in a nested and balanced fashion. That is, we must be able to match some left parenthesis against a right parenthesis that appears immediately to its right, remove both of them, and repeat. If we eventually eliminate all the parentheses, then the string was balanced, and if we cannot match parentheses in this way, then it is unbalanced. Examples of strings of balanced parentheses are (()), ()(), (()()), and ϵ, while)(and (() are not.

A grammar $G_{bal} = (\{B\}, \{(,)\}, P, B)$ generates all and only the strings of balanced parentheses, where P consists of the productions:

$$B \to BB \mid (B) \mid \epsilon$$

If-else

There is a related pattern that appears occasionally, where "parentheses" can be balanced with the exception that there can be unbalanced left parentheses. An example is the treatment of **if** and **else** in C. An if-clause can appear unbalanced by any else-clause, or it may be balanced by a matching else-clause. A grammar that generates the possible sequences of **if** and **else** (represented by i and e, respectively) is:

$$S \to \epsilon \mid SS \mid iS \mid iSeS$$

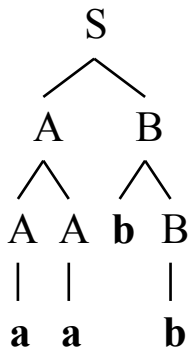
For instance, $ieie$, iie, and iei are possible sequences of **if**'s and **else**'s, and each of these strings is generated by the above grammar. Some examples of illegal sequences, not generated by the grammar, are ei and $ieeii$.

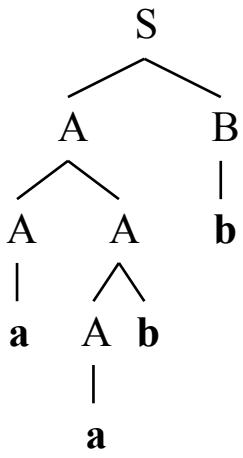
Ambiguous Grammar Derivation Trees

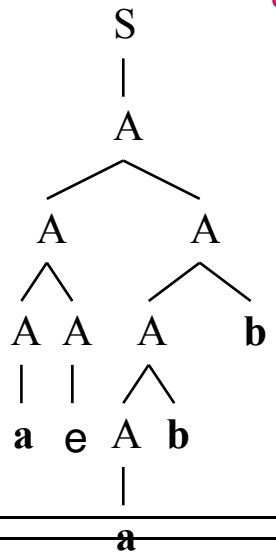
S → A | A B
A → e | **a** | A **b** | A A
B → **b** | **b c** | B **c** | **b** B

w = **aabb**

Other derivation trees for this string?







?

?

Infinitely many others possible.

Ambiguous Grammar

Definition. A CFG is *ambiguous* if there is a string in the language that has two or more parse trees

A grammar G is ambiguous if there is a word w □ L(G) having are least two different parse trees

S □ A
S □ B
S □ AB
A □ aA
B □ bB
A □ *e*
B □ *e*

Notice that a has at least two left-most derivations

Ambiguity

CFG *ambiguous* □ any of following equivalent statements:

- □ string w with multiple derivation trees.
- □ string w with multiple leftmost derivations.
- □ string w with multiple rightmost derivations.

Defining ambiguity of grammar, not language.

Ambiguity..

If there are two different parse trees, they must produce two different leftmost derivations by the construction given in the proof.

Conversely, two different leftmost derivations produce different parse trees by the other part of the proof.

Likewise for rightmost derivations.

Thus, equivalent definitions of "ambiguous grammar" are:

1. There is a string in the language that has two different leftmost derivations.
2. There is a string in the language that has two different rightmost derivations.

Example of ambiguity

Example 5.25: For instance, consider the sentential form $E + E * E$. It has two derivations from E:

1. $E \Rightarrow E + E \Rightarrow E + E * E$
2. $E \Rightarrow E * E \Rightarrow E + E * E$

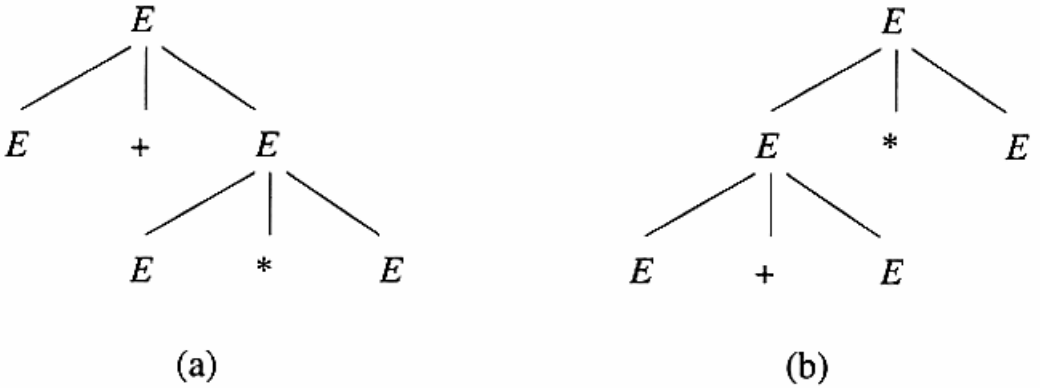


Figure 5.17: Two parse trees with the same yield

Example of ambiguity

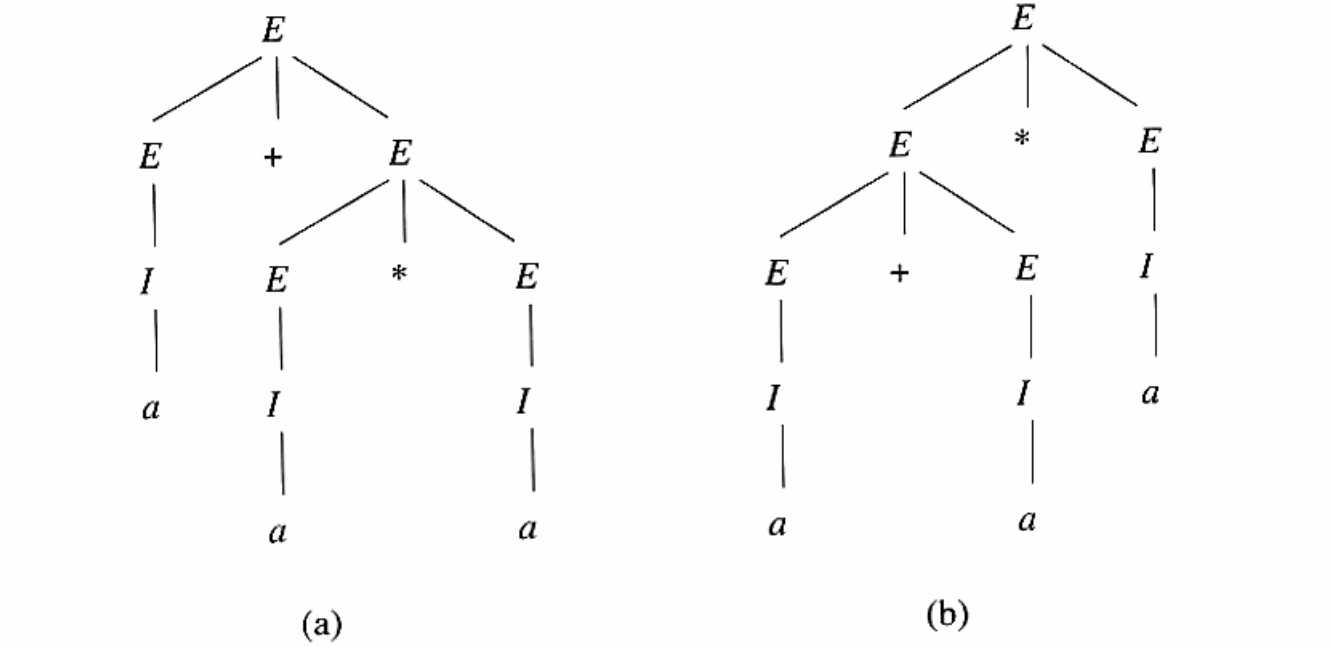


Figure 5.18: Trees with yield $a + a * a$, demonstrating the ambiguity of our expression grammar

Removing Ambiguity

There is no algorithm that can decide whether an arbitrary CFG is ambiguous, nor to remove all ambiguity

Some ambiguity can be removed by revising the CFG, such as separating the order of + and * in expressions:

There are two reasons for ambiguity in the previous slide

- the precedence of the operation is not considered.
- the sequence of identical operator can group either from the left or from the right.

Removing Ambiguity

The solution to the problem of enforcing precedence is to introduce several different variables, each of which represents those expressions that share a level of "binding strength." Specifically:

1. A *factor* is an expression that cannot be broken apart by any adjacent operator, either a $*$ or a $+$. The only factors in our expression language are:
   (a) Identifiers. It is not possible to separate the letters of an identifier by attaching an operator.
   (b) Any parenthesized expression, no matter what appears inside the parentheses. It is the purpose of parentheses to prevent what is inside from becoming the operand of any operator outside the parentheses.
2. A *term* is an expression that cannot be broken by the $+$ operator. In our example, where $+$ and $*$ are the only operators, a term is a product of one or more factors. For instance, the term $a * b$ can be "broken" if we use left associativity and place $a1*$ to its left. That is, $a1 * a * b$ is grouped $(a1 * a) * b$, which breaks apart the $a * b$. However, placing an additive term, such as $a1+$, to its left or $+a1$ to its right cannot break $a * b$. The proper grouping of $a1 + a * b$ is $a1 + (a * b)$, and the proper grouping of $a * b + a1$ is $(a * b) + a1$.

Removing Ambiguity

3. An *expression* will henceforth refer to any possible expression, including those that can be broken by either an adjacent $*$ or an adjacent $+$. Thus, an expression for our example is a sum of one or more terms.

$$
\begin{array}{lcl}
I & \rightarrow & a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
F & \rightarrow & I \mid (E) \\
T & \rightarrow & F \mid T * F \\
E & \rightarrow & T \mid E + T
\end{array}
$$

Removing Ambiguity

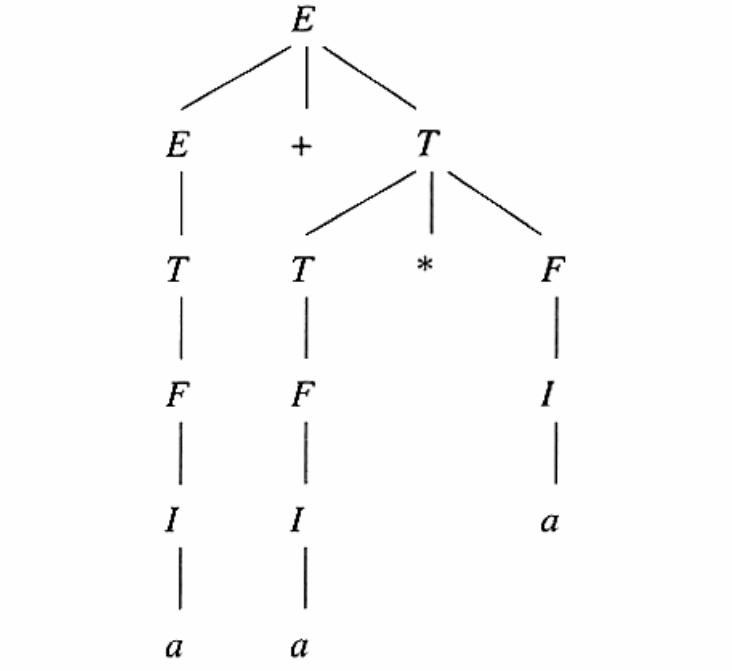


Figure 5.20: The sole parse tree for $a + a * a$

Inherently ambiguous

Unfortunately, certain CFL's are *inherently ambiguous*, meaning that every grammar for the language is ambiguous.