Ambiguity in Context Free Grammars

One sentence.
More than one syntactic structure.

Parse trees • Precedence • Associativity

Learning outcomes

Define

Recognize ambiguity in a context free grammar.

Demonstrate

Construct two distinct parse trees or leftmost derivations for a witness string.

Resolve

Rewrite expression grammars to enforce precedence and associativity.

Formal definition

Ambiguous grammar

A CFG G is ambiguous if some $w \in L(G)$ has two or more distinct parse trees.

Equivalent test

Find a sentence with two distinct leftmost derivations (or two distinct rightmost derivations).

Important

One witness is sufficient. Different orders of expansion of the same tree are not sufficient.

Example 1: Expression grammar

Grammar

$E \to E + E \mid E * E \mid id$

Witness

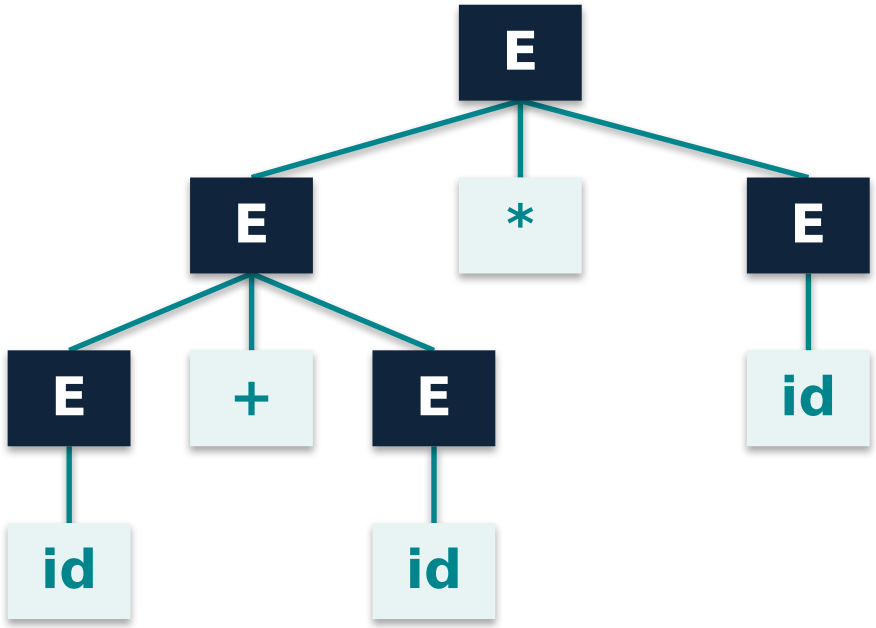
$w = id + id * id$

Two interpretations

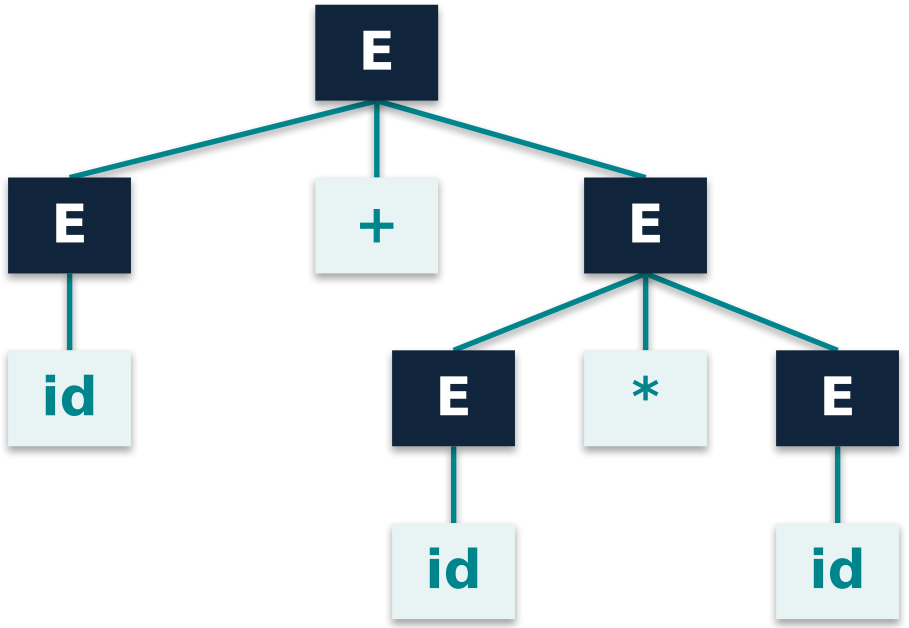
$(id + id) * id$ versus $id + (id * id)$. The grammar does not impose operator precedence.

Two parse trees for the same sentence

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



(id + id) * id



id + (id * id)

Two distinct leftmost derivations

Derivation A: multiplication at root

$$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow id + E * E$$

$$\Rightarrow id + id * E \Rightarrow id + id * id$$

Derivation B: addition at root

$$E \Rightarrow E + E \Rightarrow id + E \Rightarrow id + E * E$$

$$\Rightarrow id + id * E \Rightarrow id + id * id$$

Conclusion

Both are leftmost derivations of the same sentence; their first productions differ.

Why ambiguity matters

Meaning

With id values 2, 3 and 4: (2 + 3) * 4 = 20, but 2 + (3 * 4) = 14.

Parsing

A compiler needs a defined syntactic structure before semantic analysis and code generation.

Grammar design

Precedence specifies which operator binds tighter; associativity specifies grouping among equal-precedence operators.

Resolving operator precedence

Rewritten grammar

$E \to E + T \mid T$; $T \to T * F \mid F$

$F \to id$

Hierarchy

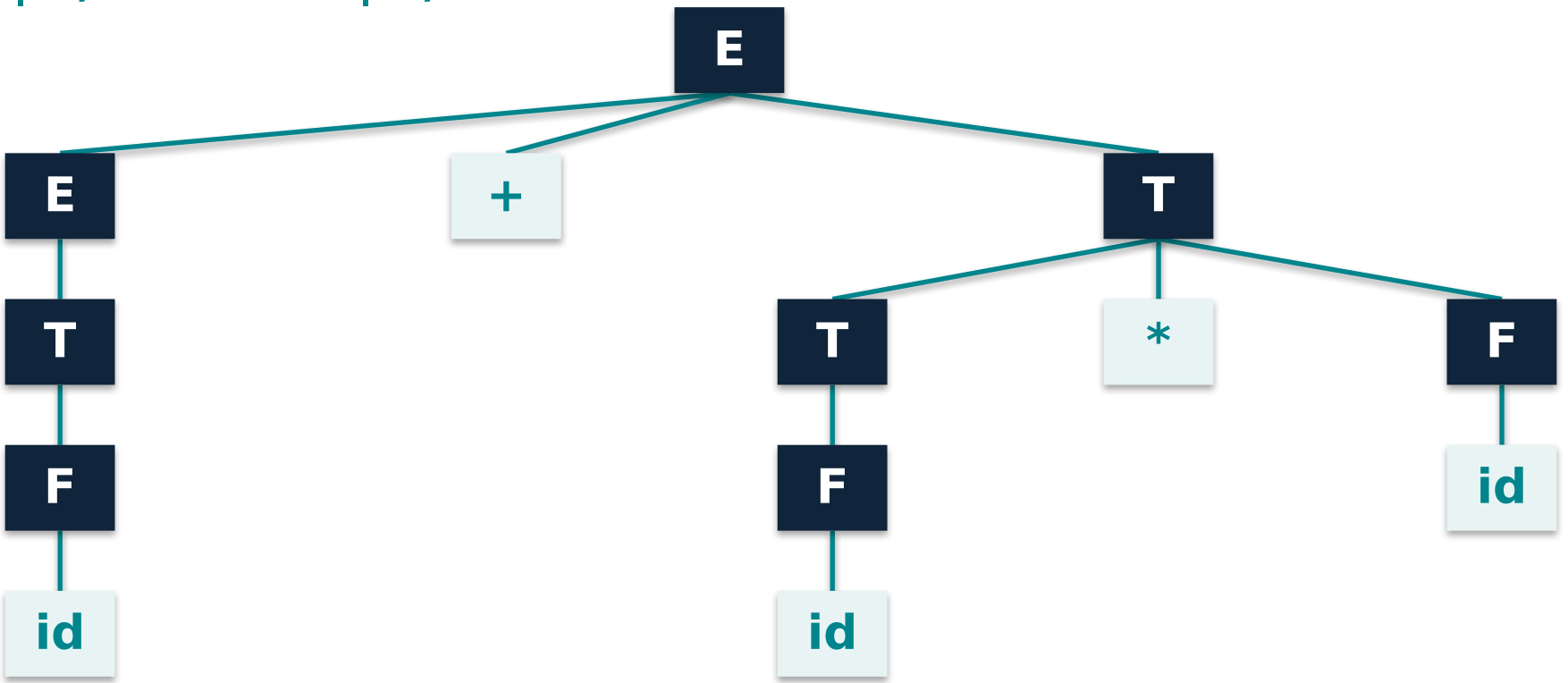
E handles addition; T handles multiplication; F handles atomic or parenthesized expressions.

Result

Multiplication binds more tightly than addition. Left recursion enforces left associativity.

Unique parse under the rewritten grammar

E → E + T | T; T → T * F | F; F → id



Structure: id + (id * id)

Example 2: Associativity

Ambiguous grammar

$E \to E - E \mid \mathrm{id}$. Witness: $\mathrm{id} - \mathrm{id} - \mathrm{id}$.

Different groupings

$(\mathrm{id} - \mathrm{id}) - \mathrm{id}$ versus $\mathrm{id} - (\mathrm{id} - \mathrm{id})$. For 8, 3, 2 the results are 3 and 7.

Left-associative grammar

$E \to E - F \mid F$; $F \to \mathrm{id}$. Left recursion forces the left grouping.

Example 3: Concatenation ambiguity

Grammar

$S \to SS \mid a$. Witness: aaa.

Two leftmost derivations

$S \Rightarrow SS \Rightarrow SSS \Rightarrow aSS \Rightarrow aaS \Rightarrow aaa$

$S \Rightarrow SS \Rightarrow aS \Rightarrow aSS \Rightarrow aaS \Rightarrow aaa$

Unambiguous replacement

$S \to aS \mid a$. Both grammars generate a^+; the replacement gives one tree per string.

Example 4: Dangling else

Grammar

S → if c then S | if c then S else S | a

Witness

if c then if c then a else a

Two interpretations

The else can attach to the inner if or the outer if. A common language rule attaches it to the nearest unmatched if.

Resolving the dangling else

Start and matched statements

S → M | U

M → a | if c then M else M

Unmatched statements

U → if c then S | if c then M else U

Why it works

A matched statement has no unmatched if. The grammar forces else to attach to the nearest eligible if.

Grammar vs. language ambiguity

Ambiguous grammar

A particular grammar may be ambiguous even if its language has another unambiguous grammar.

Inherently ambiguous language

Every CFG generating that language is ambiguous. A standard example is $\{a^i b^j c^k : i=j \text{ or } j=k,\ i,j,k \geq 1\}$.

Limitation

There is no algorithm that decides ambiguity for every CFG. Finding no witness is not a proof of unambiguity.

Classroom exercises

Exercise 1

For E → E + E | id, give two distinct leftmost derivations of id + id + id.

Exercise 2

For S → SS | b, show ambiguity using bbb and give an unambiguous equivalent grammar.

Exercise 3

Rewrite E → E + E | E * E | id to enforce precedence and left associativity.

Solutions and recap

1. Two structures

$E \Rightarrow E+E \Rightarrow E+E+E \Rightarrow id+E+E \Rightarrow id+id+E \Rightarrow id+id+id$

$E \Rightarrow E+E \Rightarrow id+E \Rightarrow id+E+E \Rightarrow id+id+E \Rightarrow id+id+id$

2. Concatenation

(bb)b and b(bb) give different trees. Replace with $S \to bS \mid b$.

3. Precedence

$E \to E+T \mid T$; $T \to T*F \mid F$; $F \to id$. A single witness proves ambiguity.