Grammars and ambiguity

CS164
3:30-5:00 TT
10 Evans

Overview

- derivations and parse trees
  - different derivations produce may produce same parse tree
- ambiguous grammars
  - what they are
  - and how to fix them

Recall: derivations and parse trees

A *derivation* is a sequence of productions

$S \to \dots \to \dots$

A derivation can be drawn as a parse tree
- Start symbol is the tree's root
- For a production $X \to Y_1 \dots Y_n$ add children $Y_1, \dots, Y_n$ to node X

You need parse trees to build ASTs

Derivation Example

- Grammar

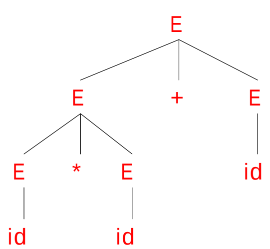
$$E \to E+E \mid E*E \mid (E) \mid id$$

- String

$$id * id + id$$

Derivation Example (Cont.)

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



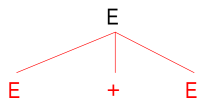
Derivation in Detail (1)

E

E

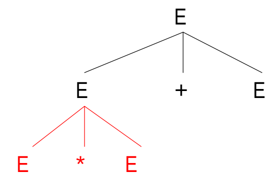
Derivation in Detail (2)

$$E$$
$$\rightarrow E+E$$



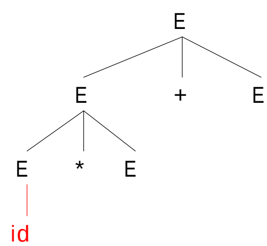
Derivation in Detail (3)

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



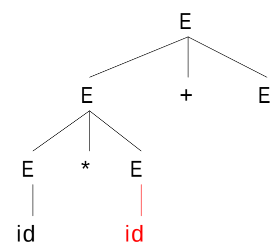
Derivation in Detail (4)

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



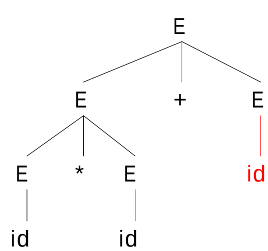
Derivation in Detail (5)

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



Derivation in Detail (6)

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



Notes on Derivations

- A parse tree has
  - Terminals at the leaves
  - Non-terminals at the interior nodes
- An in-order traversal of the leaves is the original input
- The parse tree shows the association of operations, the input string does not

Left-most and Right-most Derivations

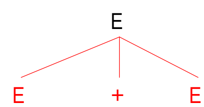
- The example is a *left-most* derivation
  - At each step, replace the left-most non-terminal
- There is an equivalent notion of a *right-most* derivation

$$
\begin{aligned}
& E \\
\rightarrow\ & E+E \\
\rightarrow\ & E+id \\
\rightarrow\ & E * E + id \\
\rightarrow\ & E * id + id \\
\rightarrow\ & id * id + id
\end{aligned}
$$

Right-most Derivation in Detail (1)

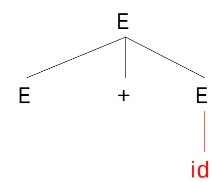
E

Right-most Derivation in Detail (2)



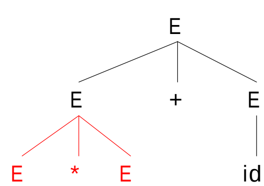
$$
\begin{aligned}
& E \\
\rightarrow\ & E+E
\end{aligned}
$$

Right-most Derivation in Detail (3)



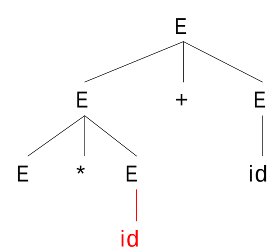
$$
\begin{aligned}
& E \\
\rightarrow\ & E+E \\
\rightarrow\ & E+id
\end{aligned}
$$

Right-most Derivation in Detail (4)



$$
\begin{aligned}
& E \\
\rightarrow\ & E+E \\
\rightarrow\ & E+id \\
\rightarrow\ & E * E + id
\end{aligned}
$$

Right-most Derivation in Detail (5)



$$
\begin{aligned}
& E \\
\rightarrow\ & E+E \\
\rightarrow\ & E+id \\
\rightarrow\ & E * E + id \\
\rightarrow\ & E * id + id
\end{aligned}
$$

Right-most Derivation in Detail (6)

E

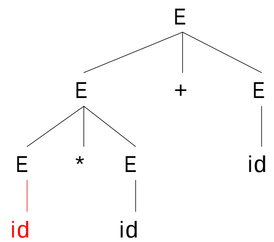
→ E+E

→ E+id

→ E ∗ E + id

→ E ∗ id + id

→ id ∗ id + id



Derivations and Parse Trees

- Note that right-most and left-most derivations have the same parse tree
- The difference is only in the order in which branches are added

ambiguity

Ambiguity

- Grammar

  E → E + E | E * E | (E) | int

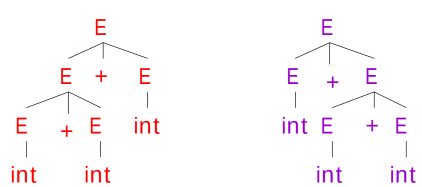
- Strings

  int + int + int

  int * int + int

Ambiguity. Example

This string has two parse trees

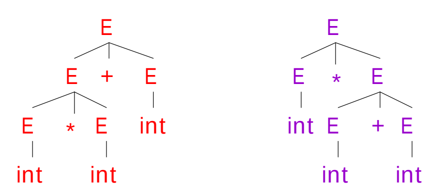


↑

+ is left-associative

Ambiguity. Example

This string has two parse trees



↑

* has higher precedence than +

Ambiguity (Cont.)

- A grammar is *ambiguous* if it has more than one parse tree for some string
  - Equivalently, there is more than one right-most or left-most derivation for some string
- Ambiguity is <u>bad</u>
  - Leaves meaning of some programs ill-defined
- Ambiguity is <u>common</u> in programming languages
  - Arithmetic expressions
  - IF-THEN-ELSE

Dealing with Ambiguity

- There are several ways to handle ambiguity
- Most direct method is to rewrite the grammar unambiguously

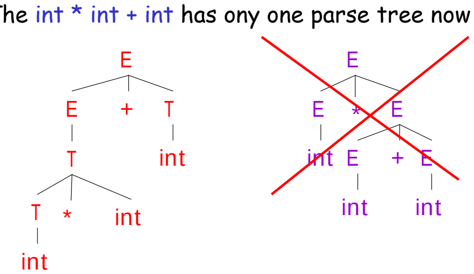
  E → E + T | T

  T → T * int | int | (E)

- Enforces precedence of * over +
- Enforces left-associativity of + and *

Ambiguity. Example

The int * int + int has ony one parse tree now



Ambiguity: The Dangling Else

- Consider the grammar

  S → if E then S

  | if E then S else S

  | OTHER

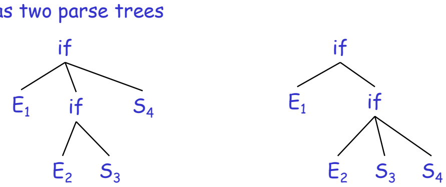
- This grammar is also ambiguous

The Dangling Else: Example

- The expression

  if E_1 then if E_2 then S_3 else S_4

  has two parse trees



- Typically we want the second form

The Dangling Else: A Fix

- else matches the closest unmatched then
- We can describe this in the grammar (distinguish between matched and unmatched "then")

  S → MIF /* all then are matched */

  | UIF /* some then are unmatched */

  MIF → if E then MIF else MIF

  | OTHER

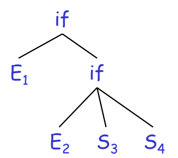
  UIF → if E then S

  | if E then MIF else UIF

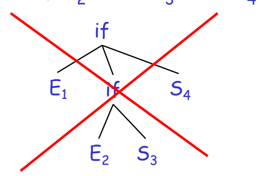
- Describes the same set of strings

The Dangling Else: Example Revisited

- The expression if E_1 then if E_2 then S_3 else S_4



- A valid parse tree (for a UIF)



- Not valid because the then expression is not a MIF

Ambiguity

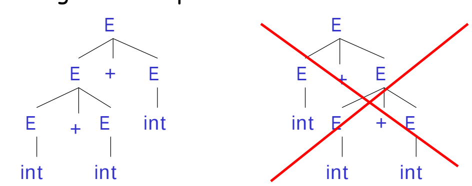
- No general techniques for handling ambiguity
- Impossible to convert automatically an ambiguous grammar to an unambiguous one
- Used with care, ambiguity can simplify the grammar
  - Sometimes allows more natural definitions
  - We need disambiguation mechanisms

Precedence and Associativity Declarations

- Instead of rewriting the grammar
  - Use the more natural (ambiguous) grammar
  - Along with disambiguating declarations
- LR (bottom-up) parsers allow <u>precedence and associativity declarations</u> to disambiguate grammars
- Examples …

Associativity Declarations

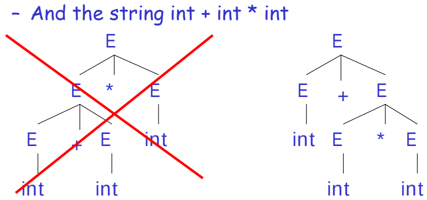
- Consider the grammar $E \rightarrow E + E \mid int$
- Ambiguous: two parse trees of int + int + int



- Left-associativity declaration: %left +

Precedence Declarations

- Consider the grammar $E \rightarrow E + E \mid E * E \mid int$
  - And the string int + int * int



- Precedence declarations: %left +
  %left *