Ambiguity

- Defining
- Rewriting:
  - Expression Grammars
    - precedence
    - associativity
  - IF-THEN-ELSE
    - the Dangling-ELSE problem
- Declarations
  - Expression Grammars
    - precedence
    - Associativity
- Readings: Sections 4.2, 4.3

Ambiguity = program structure not uniquely defined

\[ E \rightarrow E+E \mid E*E \mid (E) \mid id \]

String \( id \ast id + id \) has two parse trees:

```
  E
 /  \
E + E
 |   |
E * E
|
 id id
```
```
  E
 /  \
E * E
 |   |
 id E + E
  |   |
 id id
```
Ambiguity

- A grammar is *ambiguous* if, for any string
  - it has more than one parse tree, or
  - there is more than one right-most derivation, or
  - there is more than one left-most derivation
  (the three conditions are equivalent)

- Ambiguity is **BAD**
  - Leaves meaning of some programs ill-defined

Dealing with Ambiguity

- There are several ways to handle ambiguity
- We will discuss two of them:
  - rewriting the grammar
  - parser-generator declarations
Expression Grammars (precedence)

- Rewrite the grammar
  - use a different nonterminal for each precedence level
  - start with the lowest precedence (MINUS)

\[
E \rightarrow E - E \mid E / E \mid (E) \mid id
\]

rewrite to

\[
E \rightarrow E - E \mid T \\
T \rightarrow T / T \mid F \\
F \rightarrow id \mid (E)
\]

Example

parse tree for \(id - id / id\)
Example: Preventing Ambiguity

- **Question:** can we construct parse tree for *id-id/id* that shows the wrong precedence?

---

Associativity

- The grammar captures operator precedence, but it is still ambiguous!
  - fails to express that both subtraction and division are *left* associative;
    - e.g., 5-3-2 is equivalent to: ((5-3)-2) and *not* to: (5-(3-2))

- Example: two parse trees for the expression 5-3-2 using the grammar given above; one that correctly groups 5-3, and one that incorrectly groups 3-2
Recursion

- Grammar is **recursive** in nonterminal X if:
  - \( X \Rightarrow^+ \ldots X \ldots \)
    - \( \Rightarrow^+ \) means "in one or more steps, X derives a sequence of symbols that includes an X"
  - Grammar is **left** recursive in X if:
    - \( X \Rightarrow^+ X \ldots \)
      - in one or more steps, X derives a sequence of symbols that *starts* with an X
  - A grammar is **right** recursive in X if:
    - \( X \Rightarrow^+ \ldots X \)
      - in one or more steps, X derives a sequence of symbols that *ends* with an X

How to fix associativity

- The grammar given above is both left and right recursive in nonterminals exp and term
  - try this: write the derivation steps that show this
- To correctly express operator associativity:
  - For left associativity, use left recursion
  - For right associativity, use right recursion
- Here's the correct grammar:
  
  \[
  \begin{align*}
  E & \rightarrow E - T \mid T \\
  T & \rightarrow T / F \mid F \\
  F & \rightarrow \text{id} \mid (E)
  \end{align*}
  \]
Ambiguity: The Dangling Else

- Consider the grammar
  
  \[ E \rightarrow \text{if } E \text{ then } E \]
  
  \[ \mid \text{if } E \text{ then } E \text{ else } E \]
  
  \[ \mid \text{print} \]

- This grammar is also ambiguous

The Dangling Else: Example

- The expression
  
  \[ \text{if } E_1 \text{ then if } E_2 \text{ then } E_3 \text{ else } E_4 \]

  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

\[
E \rightarrow \text{MIF} \quad /*\text{ all then are matched */}\n| \quad \text{UIF} \quad /*\text{ some then are unmatched */}\n\]

MIF \rightarrow \text{if E then MIF else MIF}
| print
UIF \rightarrow \text{if E then E}
| \text{if E then MIF else UIF}

- Describes the same set of strings

The Dangling Else: Example Revisited

- The expression \( \text{if } E_1 \text{ then if } E_2 \text{ then } E_3 \text{ else } E_4 \)

- A valid parse tree (for a UIF)
- Not valid because the then expression is not a MIF
**Precedence and Associativity Declarations**

- Instead of rewriting the grammar
  - Use the more natural (ambiguous) grammar
  - Along with disambiguating declarations

- Most parser generators allow precedence and associativity declarations to disambiguate grammars

- Examples ...

---

**Associativity Declarations**

- Consider the grammar: \( E \to E \cdot E \mid \text{int} \)
- Ambiguous: two parse trees of \( \text{int} \cdot \text{int} \cdot \text{int} \)

- Left associativity declaration: \( \%\text{left} \ + \)
Precedence Declarations

- Consider grammar \( E \rightarrow E + E | E * E | \text{int} \)
- And the string \( \text{int} + \text{int} * \text{int} \)

- Precedence declarations: %left +
  %left *

![Precedence Diagram]

- Diagram showing the precedence of + and * operations