

Compiling Techniques

Lecture 6: Dealing with Ambiguity + Bottom-Up Parsing

Ambiguity Definition

- If a grammar has more than one leftmost (or rightmost) derivation for a single sentential form, the grammar is *ambiguous*
- This is a problem when interpreting an input program or when building an internal representation

Ambiguous Grammar: Example Associativity

Ambiguous Grammar: example 1

```
Expr ::= Expr Op Expr | num | id
Op   ::= + | *
```

This grammar has multiple leftmost derivations for $x + 2 * y$.

One possible derivation

```
Expr
Expr Op Expr
id(x) Op Expr
id(x) + Expr
id(x) + Expr Op Expr
id(x) + num(2) Op Expr
id(x) + num(2) * Expr
id(x) + num(2) * id (y)
```

$x + (2 * y)$

Another possible derivation

```
Expr
Expr Op Expr
Expr Op Expr Op Expr
id(x) Op Expr Op Expr
id(x) + Expr Op Expr
id(x) + num(2) Op Expr
id(x) + num(2) * Expr
id(x) + num(2) * id (y)
```

$(x + 2) * y$

Ambiguous Grammar: Example If-Then-Else

Ambiguous Grammar: example 2

```
Stmt ::= if Expr then Stmt  
      | if Expr then Stmt else Stmt  
      | OtherStmt
```

Input

if E1 then if E2 then S1 else S2

One possible interpretation

```
if E1 then  
  if E2 then  
    S1  
else  
  S2
```

Another possible interpretation

```
if E1 then  
  if E2 then  
    S1  
else  
  S2
```

Removing Ambiguity

- Must rewrite the grammar to avoid generating the problem
- Match each else to innermost unmatched if (common sense)

Unambiguous grammar

```
Stmt ::= if Expr then Stmt  
      | if Expr then WithElse else Stmt  
      | OtherStmt
```

```
WithElse ::= if Expr then WithElse else WithElse  
          | OtherStmt
```

- Intuition: the `WithElse` restricts what can appear in the then part
- With this grammar, the example has only one derivation

Derivation with Unambiguous Grammar

```
Stmt ::= if Expr then Stmt  
      | if Expr then WithElse else Stmt  
      | OtherStmt
```

```
WithElse ::= if Expr then WithElse else WithElse  
          | OtherStmt
```

Derivation for: if E1 then if E2 then S1 else S2

```
Stmt  
if Expr then Stmt  
if E1 then Stmt  
if E1 then if Expr then WithElse else Stmt  
if E1 then if E2 then WithElse else Stmt  
if E1 then if E2 then S1 else Stmt  
if E1 then if E2 then S1 else S2
```

Deeper Ambiguity

- Ambiguity usually refers to confusion in the CFG (Context Free Grammar)
- Consider the following case: $a = f(17)$
In Algol-like languages, f could be either a function or an array
- In such case, context is required
 - Need to track declarations
 - Really a type issue, not context-free syntax
 - Requires an extra-grammatical solution
 - Must handle these with a different mechanism

Step outside the grammar rather than making it more complex. This will be treated during semantic analysis.

Ambiguity Final Words

Ambiguity arises from two distinct sources:

- Confusion in the context-free syntax (e.g. *if then else*)
- Confusion that requires context to be resolved (e.g. *array vs function*)

Resolving ambiguity:

- To remove context-free ambiguity, rewrite the grammar
- To handle context-sensitive ambiguity delay the detection of such problem (semantic analysis phase):

For instance, it is legal during syntactic analysis to have: `void i ; i=4;`

Bottom-Up vs. Top-Down Parsers

Top-Down Parser

A top-down parser builds a derivation by working from the start symbol to the input sentence.



Bottom-Up Parser

A bottom-up parser builds a derivation by working from the input sentence back to the start symbol.



Bottom-Up Parsing: Example

Example: CFG

Goal ::= a A B e

A ::= A b c | b

B ::= d

Input: abcde

Bottom-Up Parsing

abcde

Bottom-Up Parsing: Example

Example: CFG

Goal ::= a A B e
A ::= A b c | b
B ::= d

Input: abcde

Bottom-Up Parsing

abcde
aAbcde

Bottom-Up Parsing: Example

Example: CFG

```
Goal ::= a A B e  
A ::= A b c | b  
B ::= d
```

Input: abcde

Bottom-Up Parsing

```
abcde  
aAbcde  
aAde
```

Bottom-Up Parsing: Example

Example: CFG

Goal ::= a A B e
A ::= A b c | b
B ::= d

Input: abcde

Bottom-Up Parsing

abcde
aAbcde
aA**d**e
aABe

Bottom-Up Parsing: Example

Example: CFG

Goal ::= a A B e

A ::= A b c | b

B ::= d

Input: abcde

Bottom-Up Parsing

productions
(follow **rightmost
derivation**)



abcde
aAbcde
aAde
aABe
Goal



reductions

Leftmost vs. Rightmost derivation

Leftmost derivation

Rewrite leftmost
nonterminal next

Example: CFG

Goal ::= a A B e
A ::= A b c | b
B ::= d

Rightmost derivation

Rewrite rightmost
nonterminal next

Leftmost derivation LL Parser (Top-Down)

Goal
aA**B**e
aA**b**cBe
ab**b**cBe
ab**b**cde

Rightmost derivation LR Parser (Bottom-Up)

Goal
aA**B**e
aA**d**e
aA**b**cde
ab**b**cde

Shift-reduce parser

Consists of a stack and the input

Uses four actions:

1. **shift**: next symbol is shifted onto the stack
2. **reduce**: pop the symbols Y_n, \dots, Y_1 from the stack that form the rhs of a production rule $X ::= Y_n, \dots, Y_1$
3. **accept**: stop parsing and report success
4. **error**: reporting an error

How does the parser know when to shift or when to reduce?

Similarly to the top-down parser, can back-track if wrong decision made or try to look ahead.
Can build a DFA to decide when to shift or to reduce.

Shift-reduce parser: Example

Input

abbcde
bbcde
bcde
bcde
cde
de
de
e
e

Operations

shift
shift
reduce
shift
shift
reduce
shift
reduce
shift
reduce
accept

Stack

a
ab
aA
aAb
aAbc
aA
aAd
aAB
aABe
Goal

Example: CFG

Goal ::= a A B e
A ::= A b c | b
B ::= d

Choice here: shift or reduce?

Can lookahead one symbol to make decision.

(Knowing what to do needs analysis of the grammar, see *Engineering a Compiler* §3.5)

Top-Down vs Bottom-Up Parsing

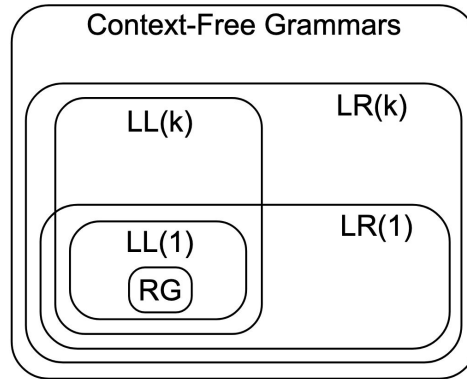
Top-Down Parser

- + Easy to write by hand
- + Easy to integrate with rest of the compiler
- Recursion might lead to performance problems

Bottom-Up Parser

- + Very efficient
- + Supports a larger class of grammars
- Requires generation tools
- Rigid integration with the rest of the compiler

Last words on Parsing



Language \neq Grammar

There is more than one grammar that can be used to define a language

These grammars might be of different “complexity” (LL(1), LL(k), LR(k))

\Rightarrow Language complexity \neq grammar complexity