Introduction to Parsing
Part II

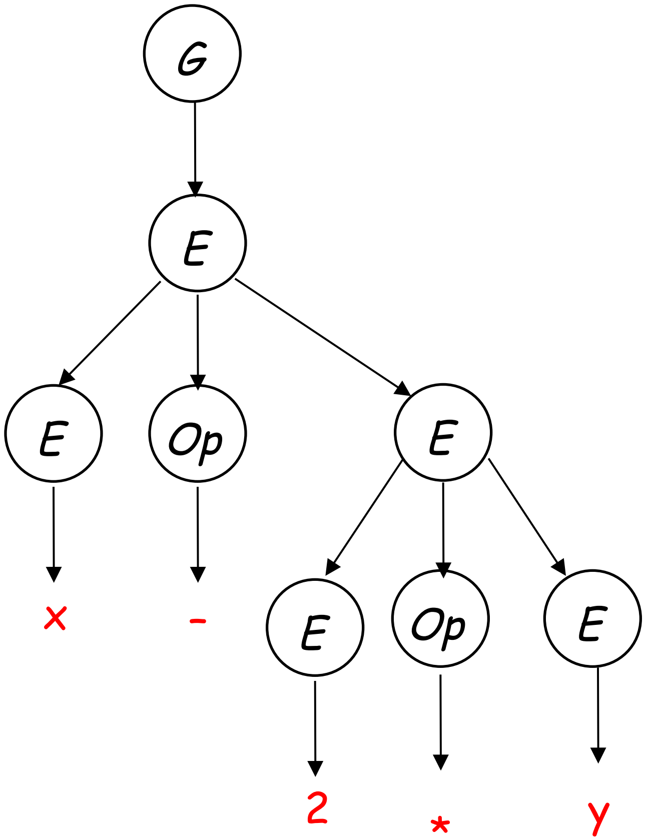
Quiz

1	*Expr*	→	*Expr Op Expr*
2		|	<u>number</u>
3		|	<u>id</u>
4	*Op*	→	+
5		|	-
6		|	*
7		|	/

Produce a table showing the **<u>rightmost derivation</u>** for the equation below. Include in the first column the rule used and the second column the sentential form.

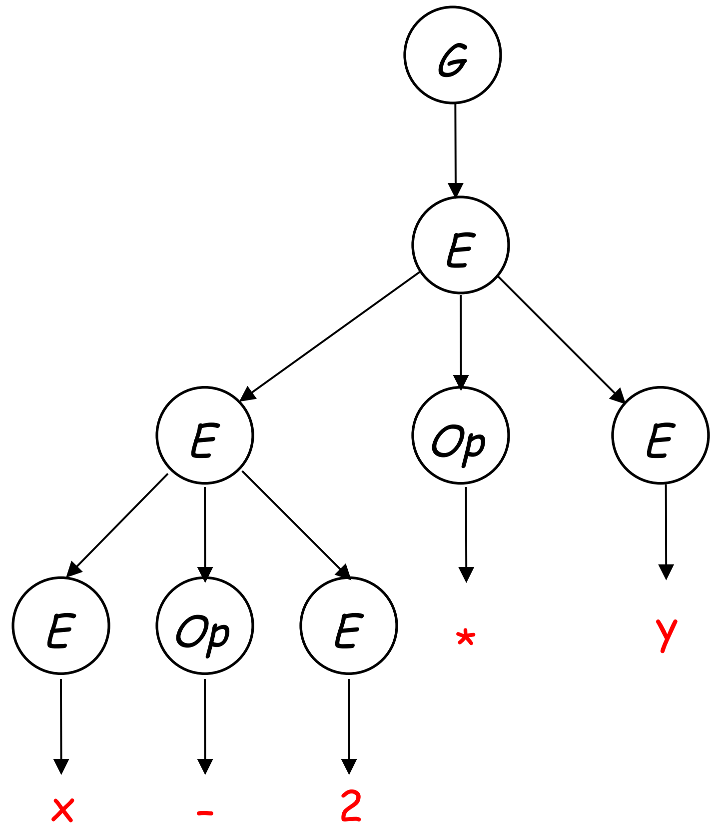
$$a * b - c$$

Leftmost derivation and Rightmost derivation



Leftmost derivation

This evaluates as x - (2 * y)



Rightmost derivation

This evaluates as (x - 2) * y

Derivations and Precedence

These two derivations point out a problem with the grammar:

We same parse tree regardless of rightmost or leftmost derivation

No notion of precedence in grammar

Key: Create a non-terminal (NT) for each *level of precedence*

Derivations and Precedence

To add precedence

- Create a non-terminal for each *level of precedence*
- Isolate the corresponding part of the grammar
- Force the parser to recognize high precedence subexpressions first

a + b – c * d

Should recognize c*d first!

For algebraic expressions

- Multiplication and division, first (*level one*)
- Subtraction and addition, next (*level two*)

Derivations and Precedence

Adding the standard algebraic precedence produces:

1	*Goal*	→	*Expr*
2	*Expr*	→	*Expr + Term*
3		|	*Expr - Term*
4		|	*Term*
5	*Term*	→	*Term * Factor*
6		|	*Term / Factor*
7		|	*Factor*
8	*Factor*	→	number
9		|	id

level two (rules 2–4)

level one (rules 5–7)

Note that you can only get to Term through Expr!

This grammar is slightly larger

- Takes more rewriting to reach some of the terminal symbols
- Encodes expected precedence
- Produces same parse tree under leftmost & rightmost derivations

*Let's see how it parses x - 2 * y*

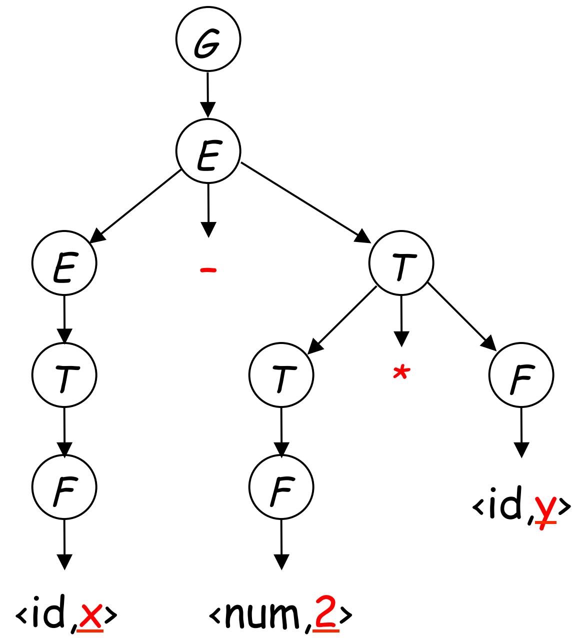
Rightmost derivation of x-2*y.

	1	*Goal*	→	*Expr*
level two	2	*Expr*	→	*Expr + Term*
	3		|	*Expr - Term*
	4		|	*Term*
level one	5	*Term*	→	*Term * Factor*
	6		|	*Term / Factor*
	7		|	*Factor*
	8	*Factor*	→	<u>number</u>
	9		|	<u>id</u>

Derivations and Precedence

Rule	*Sentential Form*
—	*Goal*
1	*Expr*
3	*Expr* - *Term*
5	*Expr* - *Term* * *Factor*
9	*Expr* - *Term* * <id,y>
7	*Expr* - *Factor* * <id,y>
8	*Expr* - <num,2> * <id,y>
4	*Term* - <num,2> * <id,y>
7	*Factor* - <num,2> * <id,y>
9	<id,x> - <num,2> * <id,y>

The rightmost derivation



Its parse tree

This produces x – (2 * y), along with an appropriate parse tree.

Both the leftmost and rightmost derivations give the same expression, because the grammar directly encodes the desired precedence.

Ambiguous Grammars

Our original expression grammar had other problems

- Let's look at original leftmost derivation

1	*Expr*	→	*Expr Op Expr*
2		|	number
3		|	id
4	*Op*	→	+
5		|	-
6		|	*
7		|	/

Rule	*Sentential Form*
—	*Expr*
1	*Expr Op Expr*
3	‹id,x› *Op Expr*
5	‹id,x› - *Expr*
1	‹id,x› - *Expr Op Expr*
2	‹id,x› - ‹num,2› *Op Expr*
6	‹id,x› - ‹num,2› * *Expr*
3	‹id,x› - ‹num,2› * ‹id,y›

Make note of the second rule we use!

Ambiguous Grammars

Our original expression grammar had other problems

- The grammar is *ambiguous*

1	*Expr*	→	*Expr Op Expr*
2		|	number
3		|	id
4	*Op*	→	+
5		|	-
6		|	*
7		|	/

Rule	*Sentential Form*
—	*Expr*
1	*Expr Op Expr*
1	*Expr Op Expr Op Expr*
3	<id,x> *Op Expr Op Expr*
5	<id,x> - *Expr Op Expr*
2	<id,x> - <num,2> *Op Expr*
6	<id,x> - <num,2> * *Expr*
3	<id,x> - <num,2> * <id,y>

different choice than the first time

Two Leftmost Derivations for *x - 2 * y*

The Difference:

- Different productions chosen on the second step
- Both derivations succeed in producing *x - 2 * y*

Rule	Sentential Form
—	*Expr*
1	*Expr Op Expr*
3	<id,x> *Op Expr*
5	<id,x> - *Expr*
1	<id,x> - *Expr Op Expr*
2	<id,x> - <num,2> *Op Expr*
6	<id,x> - <num,2> * *Expr*
3	<id,x> - <num,2> * <id,y>

Original choice

Rule	Sentential Form
—	*Expr*
1	*Expr Op Expr*
1	*Expr Op Expr Op Expr*
3	<id,x> *Op Expr Op Expr*
5	<id,x> - *Expr Op Expr*
2	<id,x> - <num,2> *Op Expr*
6	<id,x> - <num,2> * *Expr*
3	<id,x> - <num,2> * <id,y>

New choice

Ambiguous Grammars

Definitions

- If a grammar has more than one leftmost derivation for a single *sentential form*, the grammar is *ambiguous*
- If a grammar has more than one rightmost derivation for a single sentential form, the grammar is *ambiguous*
- The leftmost and rightmost derivations for a sentential form may differ, even in an unambiguous grammar

If-then-else problem

Classic example

Stmt → if *Expr* then *Stmt*
| if *Expr* then *Stmt* else *Stmt*
| … *other stmts* …

This ambiguity is entirely grammatical in nature

Ambiguity

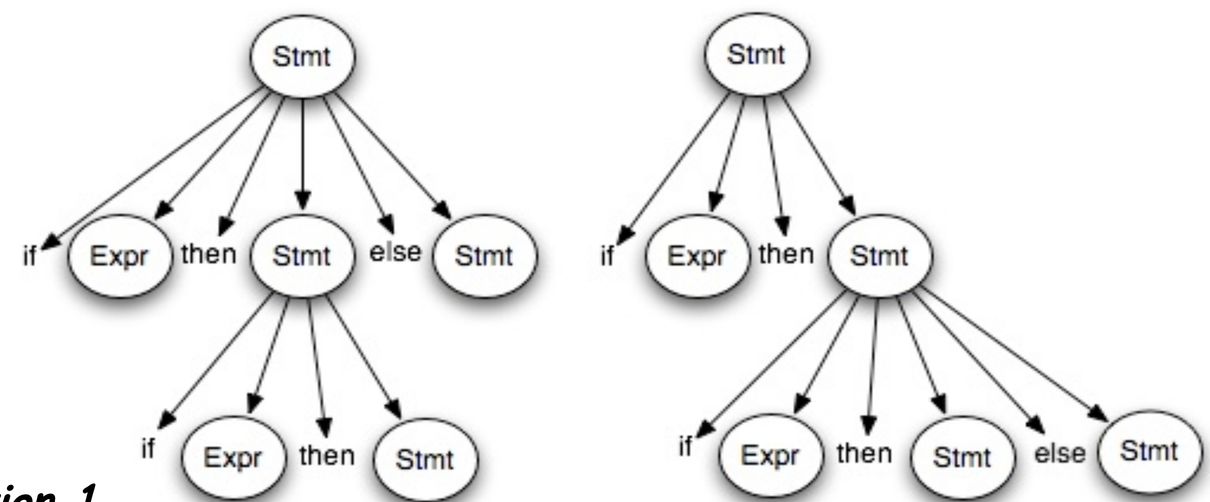
This if statement has two derivations

if $Expr_1$ then if $Expr_2$ then $Stmt_1$ else $Stmt_2$

Stmt → if *Expr* then *Stmt* (1)
| if *Expr* then *Stmt* else *Stmt* (2)
| … *other stmts* …

production 2

then production 1



production 1,

then production 2

Ambiguity

Removing the ambiguity

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

With this grammar, the example has only one derivation

1	*Statement*	→	if *Expr* then *Statement*
2		|	if *Expr* then *WithElse* else *Statement*
3		|	*Assignment*
4	*WithElse*	→	if *Expr* then *WithElse* else *WithElse*
5		|	Assignment

Intuition: binds each else to the innermost if

Ambiguity

<u>   if</u> $Expr_1$ <u>then</u> <u>if</u> $Expr_2$ <u>then</u> $Assignment_1$ <u>else</u> $Assignment_2$

1	*Statement*	→	<u>if</u> *Expr* <u>then</u> *Statement*
2		|	<u>if</u> *Expr* <u>then</u> *WithElse* <u>else</u> *Statement*
3		|	*Assignment*
4	*WithElse*	→	<u>if</u> *Expr* <u>then</u> *WithElse* <u>else</u> *WithElse*
5		|	*<u>Assignment</u>*

Rule	*Sentential Form*
	Statement
1	<u>if</u> *Expr* <u>then</u> *Statement*
2	<u>if</u> *Expr* <u>then</u> <u>if</u> *Expr* <u>then</u> *WithElse* <u>else</u> *Statement*
3	<u>if</u> *Expr* <u>then</u> <u>if</u> *Expr* <u>then</u> *WithElse* <u>else</u> *Assignment*
5	<u>if</u> *Expr* <u>then</u> <u>if</u> *Expr* <u>then</u> *Assignment* <u>else</u> *Assignment*

This binds the <u>else</u> controlling $Assignment_2$ to the inner <u>if</u>

Deeper Ambiguity

Ambiguity usually refers to confusion in the CFG

Overloading can create deeper ambiguity

$$a = f(17)$$

In many Algol-like languages, f could be either a function or a subscripted variable

Disambiguating this one requires **context**

- Need values of declarations
- Really an issue of *type*, not context-free syntax
- Must handle these with a different mechanism

Ambiguity - the Final Word

Ambiguity arises from two distinct sources

- Confusion in the context-free syntax (*if-then-else*)
- Confusion that requires context to resolve (*overloading*)

Resolving ambiguity

- To remove context-free ambiguity, rewrite the grammar
- To handle context-sensitive ambiguity takes cooperation
  - → Knowledge of declarations, types, …
  - → Accept a superset of *L(G)* & check it by other means†

†See Chapter 4