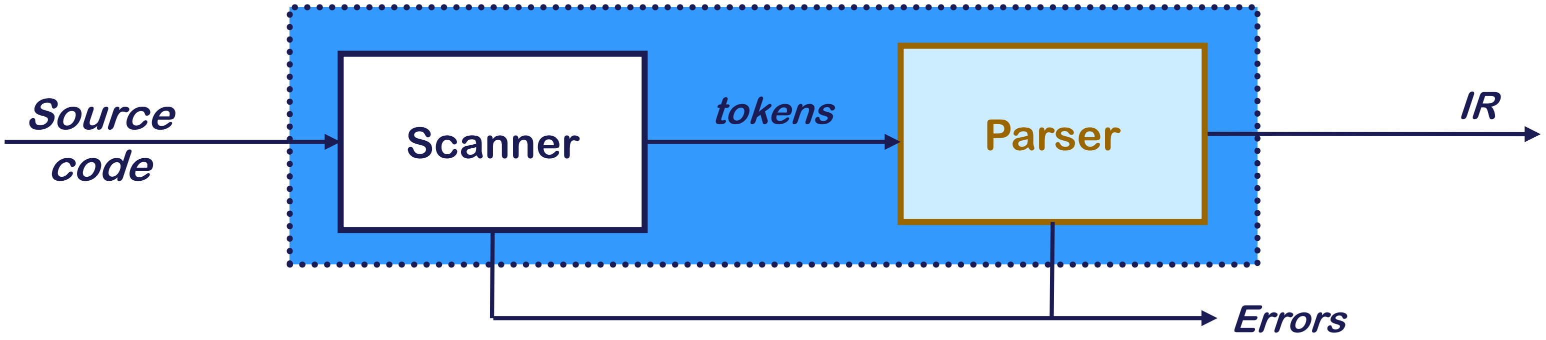
Programming Language & Compiler

Context-Free Grammar

Hwansoo Han

Parser in Front End



- **Parser**
  - Checks the stream of <u>words</u> and their <u>parts of speech</u> for grammatical correctness
  - Determines if the input is syntactically well formed
  - Guides checking at deeper levels than syntax
  - Builds an IR representation of the code

The Study of Parsing

- **The process of discovering a *derivation* for some sentence**
  - Need a mathematical model of syntax — a grammar *G*
  - Need an algorithm for testing membership in *L(G)*
  - Need to keep in mind that our goal is building parsers, not studying the mathematics of arbitrary languages

- **Roadmap**
  1. Context-free grammars and derivations
  2. Top-down parsing
  3. Bottom-up parsing

Specification of Grammar

- **Syntax is specified with CFG = <S, T, N, P>**

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>

- Such a sequence of rewrites is called a *derivation*
- Process of discovering a derivation is called *parsing*

We denote this derivation: *Expr* $\Rightarrow^*$ id - num * id

Derivations

- **Derivation consists of multiple steps of rewrites**
  - At each step, we choose a non-terminal to replace
  - Different choices can lead to different derivations
- **Two derivations are of interest**
  - *Leftmost derivation* — replace leftmost NT at each step
  - *Rightmost derivation* — replace rightmost NT at each step
  - These are the two *systematic* derivations
    (We don't care about randomly-ordered derivations!)
- **The example on the preceding slide was a *leftmost* derivation**
  - Of course, there is also a *rightmost* derivation
  - Interestingly, it turns out to be different

The Two Derivations for 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>

Leftmost derivation

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

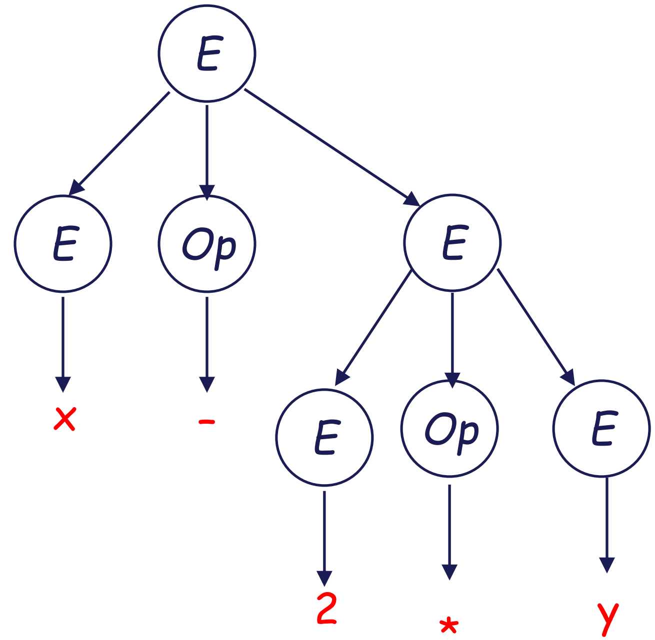
Rightmost derivation

- **In both cases, *Expr* $\Rightarrow^*$ id - num * id**
  - The two derivations produce different parse trees
  - The parse trees imply different evaluation orders!

Derivations and Parse Trees (1)

Leftmost derivation

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>

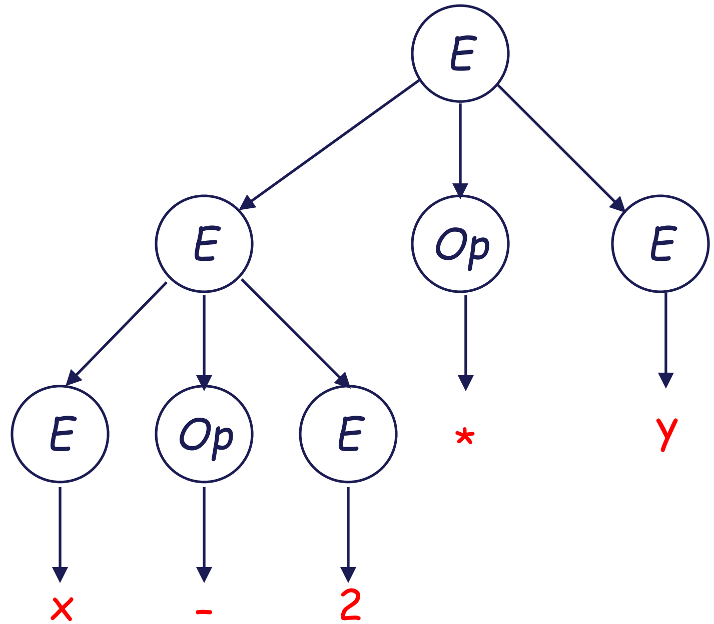


This evaluates as x - (2 * y)

Derivations and Parse Trees (2)

Rightmost derivation

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



This evaluates as (x – 2) * y

Reduction

- **Rightmost derivation requires backward scan**
  - In reality, we can scan from the left and apply derivation in a reverse way
- **Reduction**
  - Reverse process of derivation
  - Production rule: $A \longrightarrow \underline{a}\beta$
    - Derivation: replace A with $\underline{a}\beta$
    - Reduction: replace $\underline{a}\beta$ with A

$$Expr \Rightarrow Expr\ Op\ \mathrm{y} \Rightarrow Expr\ Op\ Expr * \mathrm{y} \Rightarrow \mathrm{x} - 2 * \mathrm{y}$$

Reduction

Precedence in Derivations (1)

- ***These two derivations point out a problem with the grammar:***
  - *It has no notion of precedence, or implied order of evaluation*

- **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

- **For algebraic expressions**
  - Multiplication and division, first : *level one* – higher precedence
  - Subtraction and addition, next : *level two* – lower precedence

Precedence in Derivations (2)

- **Adding the standard algebraic precedence produces:**

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

This grammar is slightly larger

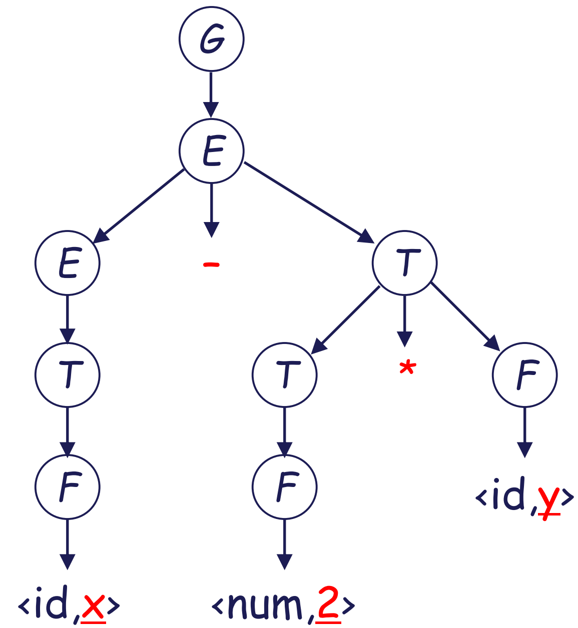
- Takes more rewriting to reach some of the terminal symbols
- Encodes expected precedence
- Produces same parse tree under leftmost & rightmost derivations

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

Precedence in Derivations (3)

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

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

- **Our original expression grammar had other problems**
  - This grammar allows multiple leftmost derivations for x - 2 * y
  - Hard to automate derivation if #choices > 1
  - Both derivations succeed in producing *x - 2 * y*

Ambiguous Grammars

- **Definitions**
  - A grammar G is *ambiguous*, if and only if there exists a single sentence in *L(G)* that has multiple rightmost (or leftmost) derivations

  - The leftmost and rightmost derivations for a sentence may differ, even in an unambiguous grammar (precedence problem)

- **Classic example — the *if-then-else* problem**

$$
\begin{array}{lll}
\mathit{Stmt} & \rightarrow & \underline{\text{if}}\ \ \mathit{Expr}\ \ \underline{\text{then}}\ \mathit{Stmt} \\
 & | & \underline{\text{if}}\ \ \mathit{Expr}\ \ \underline{\text{then}}\ \mathit{Stmt}\ \ \underline{\text{else}}\ \ \mathit{Stmt} \\
 & | & \ldots\ \mathit{other\ stmts}\ \ldots
\end{array}
$$

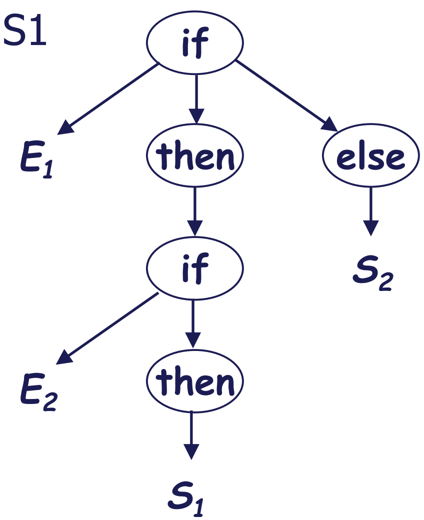
  - *This ambiguity is entirely grammatical in nature*

Ambiguity

- **This sentential form has two derivations**

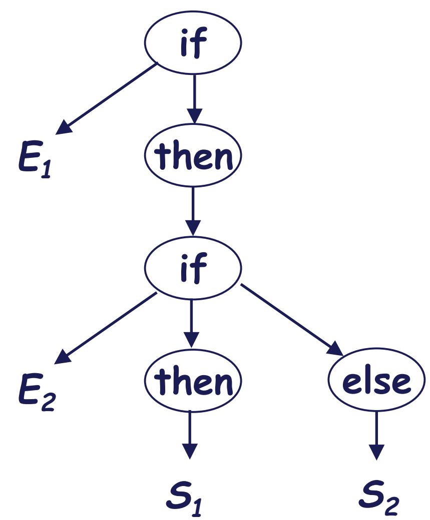
  if *E1* then if *E2* then *S1* else *S2*

if E1
  then if E2
        then S1
  else S2



production 2, then production 1

if E1
  then if E2
        then S1
        else S2



production 1 , then production 2

Ambiguity

Removing the ambiguity

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

1	*Stmt*	$\rightarrow$	*WithElse*
2		|	*NoElse*
3	*WithElse*	$\rightarrow$	<u>if</u> *Expr* <u>then</u> *WithElse* <u>else</u> *WithElse*
4		|	*OtherStmt*
5	*NoElse*	$\rightarrow$	<u>if</u> *Expr* <u>then</u> *Stmt*
6		|	<u>if</u> *Expr* <u>then</u> *WithElse* <u>else</u> *NoElse*

<u>Intuition:</u>
Between <u>then</u> and <u>else</u>, only *WithElse* can go, but *NoElse cannot.*

With this grammar, the example has only one derivation

Ambiguity

- **if E_1 then if E_2 then S_1 else S_2**

Rule	Sentential Form
—	*Stmt*
2	*NoElse*
5	if *Expr* then *Stmt*
?	if E_1 then *Stmt*
1	if E_1 then *WithElse*
3	if E_1 then if *Expr* then *WithElse* else *WithElse*
?	if E_1 then if E_2 then *WithElse* else *WithElse*
4	if E_1 then if E_2 then S_1 else *WithElse*
4	if E_1 then if E_2 then S_1 else S_2

This binds the else controlling S_2 to the inner if

Resolve If-Then-Else with Precedence

- **Precedence enforces which operation to apply first**
  - If we have choices between If-Then and If-Then-Else
    - apply If-Then first (lower precedence) in derivation
    - apply If-Then-Else first (higher precedence) in reduction

- **if E_1 then if E_2 then S_1 else S_2**
  - When we need to reduce for

    if E_2 then S_1 else S_2

    choose If-Then-Else instead of If-Then

$$\text{if } E_1 \text{ then } \textit{Statement} \underset{\xleftarrow{\text{reduction}}}{\Rightarrow} \text{if } E_1 \text{ then } \boxed{\text{if } E_2 \text{ then } S_1 \text{ else } S_2}$$

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 (i.e. array access)

- **Disambiguating this one requires context**
  - Need values of declarations
  - Really an issue of *type*, not context-free syntax
  - Requires an extra-grammatical solution (not in CFG)
  - Must handle these with a different mechanism
    - Step outside grammar rather than use a more complex grammar
    - Context-sensitive analysis

Summary

- **Derivation**
  - Leftmost derivation or rightmost derivation
  - Precedence is needed to get intended parse-tree
  - Two more derivations $\Rightarrow$ ambiguous grammar