Context-Free Grammars

Using grammars in parsers

Jaruloj Chongstitvatana
Department of Mathematics and Computer Science
Chulalongkorn University

Outline

- Parsing Process
- Grammars
  - Context-free grammar
  - Backus-Naur Form (BNF)
- Parse Tree and Abstract Syntax Tree
- Ambiguous Grammar
- Extended Backus-Naur Form (EBNF)

Parsing Process

- Call the scanner to get tokens
- Build a parse tree from the stream of tokens
  - A parse tree shows the syntactic structure of the source program.
- Add information about identifiers in the symbol table
- Report error, when found, and recover from thee error

Grammar

- a quintuple (*V*, *T*, *P*, *S*) where
  - *V* is a finite set of nonterminals, containing *S*,
  - *T* is a finite set of terminals,
  - *P* is a set of production rules in the form of $\alpha \rightarrow \beta$ where α and β are strings over $V \cup T$, and
  - *S* is the start symbol.

หนึ่งตัวเดียว non terminal

- Example

G= ({S, A, B, C}, {a, b, c}, P, S)

P= { S→SABC, BA → AB, CB → BC, CA → AC,

SA → a, aA → aa, aB → ab, bB → bb,
bC → bc, cC → cc}

Context-Free Grammar

- a quintuple (*V, T, P, S*) where
  - *V* is a finite set of nonterminals, containing *S*,
  - *T* is a finite set of terminals,
  - *P* is a set of production rules in the form of $\alpha \rightarrow \beta$ where α is in *V* and β is in $(V \cup T)^*$, and
  - *S* is the start symbol.
- Any string in (V U T)* is called a *sentential form*.

Examples

$E \rightarrow E\ O\ E$

$E \rightarrow (E)$

$E \rightarrow id$

$O \rightarrow +$

$O \rightarrow -$

$O \rightarrow *$

$O \rightarrow /$

$S \rightarrow SS$

$S \rightarrow (S)S$

$S \rightarrow ()$

$S \rightarrow \lambda$

Backus-Naur Form (BNF)

- Nonterminals are in < >.
- Terminals are any other symbols.
- ::= means $\rightarrow$.
- | means or.
- Examples:

  <E> ::= <E><O><E>| (<E>) | ID

  <O> ::= + | - | * | /

  <S> ::= <S><S> | (<S>)<S> | () | λ

Derivation

- A sequence of replacement of a substring in a sentential form.

Definition

- Let $G=(V, T, P, S)$ be a CFG, α, β, γ be strings in $(V \cup T)^*$ and A is in V.

  $\alpha A\beta \Rightarrow_G \alpha\gamma\beta$ if $A \rightarrow \gamma$ is in P.

- $\Rightarrow^*_G$ denotes a derivation in zero step or more.

Examples

$S \to SS \mid (S)S \mid () \mid \lambda$

S
$\Rightarrow SS$
$\Rightarrow (S)SS$
$\Rightarrow (S)S(S)S$
$\Rightarrow (S)S(())S$
$\Rightarrow ((S)S)S(())S$
$\Rightarrow ((S)())S(())S$
$\Rightarrow ((())())S(())S$
$\Rightarrow ((())())\ (())S$
$\Rightarrow ((())())(())$

$E \to E\ O\ E \mid (E) \mid id$
$O \to + \mid - \mid * \mid /$

E
$\Rightarrow E\ O\ E$
$\Rightarrow (E)\ O\ E$
$\Rightarrow (E\ O\ E)\ O\ E$
$\Rightarrow^* ((E\ O\ E)\ O\ E)\ O\ E$
$\Rightarrow ((id\ O\ E))\ O\ E)\ O\ E$
$\Rightarrow ((id + E))\ O\ E)\ O\ E$
$\Rightarrow ((id + id))\ O\ E)\ O\ E$
$\Rightarrow^* ((id + id)) * id) + id$

Leftmost Derivation Rightmost Derivation

- Each step of the derivation is a replacement of the *leftmost* nonterminals in a sentential form.

E
⇒ **E** O E
⇒ (**E**) O E
⇒ (**E** O E) O E
⇒ (id **O** E) O E
⇒ (id + **E**) O E
⇒ (id + id) **O** E
⇒ (id + id) * **E**
⇒ (id + id) * id

- Each step of the derivation is a replacement of the *rightmost* nonterminals in a sentential form.

E
⇒ E O **E**
⇒ E **O** id
⇒ **E** * id
⇒ (**E**) * id
⇒ (E O **E**) * id
⇒ (E **O** id) * id
⇒ (**E** + id) * id
⇒ (id + id) * id

Language Derived from Grammar

- Let $G = (V, T, P, S)$ be a CFG.
- A string w in T^* is derived from G if $S \stackrel{*}{\Rightarrow}_G w$.
- A language generated by G, denoted by $L(G)$, is a set of strings derived from G.
  - $L(G) = \{w \mid S \stackrel{*}{\Rightarrow}_G w\}$.

Right/Left Recursive

- A grammar is a *left recursive* if its production rules can generate a derivation of the form $A \Rightarrow^* A\ X$.
- Examples:
  - $E \rightarrow E\ O\ id \mid (E) \mid id$
  - $E \rightarrow F + id \mid (E) \mid id$

    $F \rightarrow E * id \mid id$
    - $E \Rightarrow F + id$

      $\Rightarrow E * id + id$

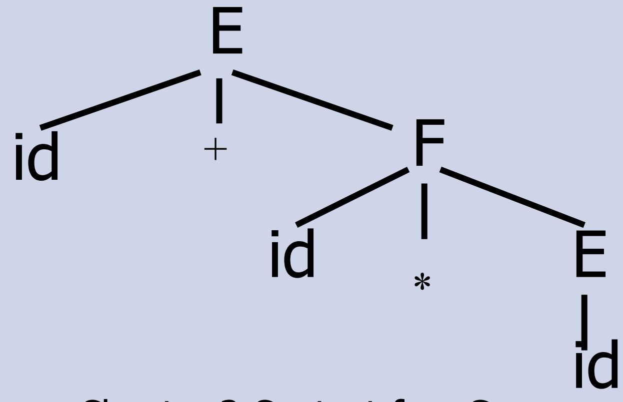
- A grammar is a *right recursive* if its production rules can generate a derivation of the form $A \Rightarrow^* X\ A$.
- Examples:
  - $E \rightarrow id\ O\ E \mid (E) \mid id$
  - $E \rightarrow id + F \mid (E) \mid id$

    $F \rightarrow id * E \mid id$
    - $E \Rightarrow id + F$

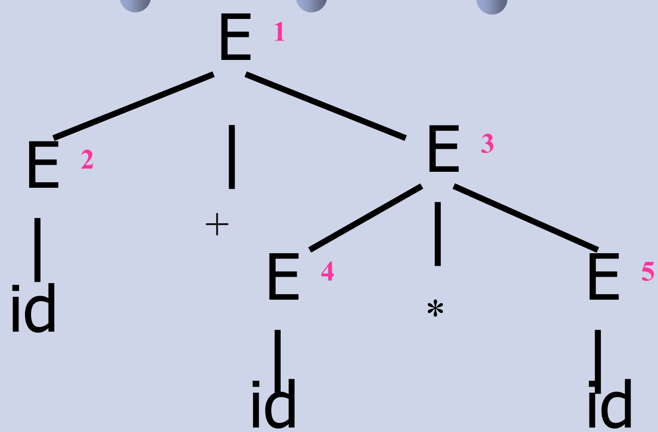
      $\Rightarrow id + id * E$

Parse Tree

- A labeled tree in which
  - the interior nodes are labeled by nonterminals
  - leaf nodes are labeled by terminals
  - the children of an interior node represent a replacement of the associated nonterminal in a derivation
  - corresponding to a derivation

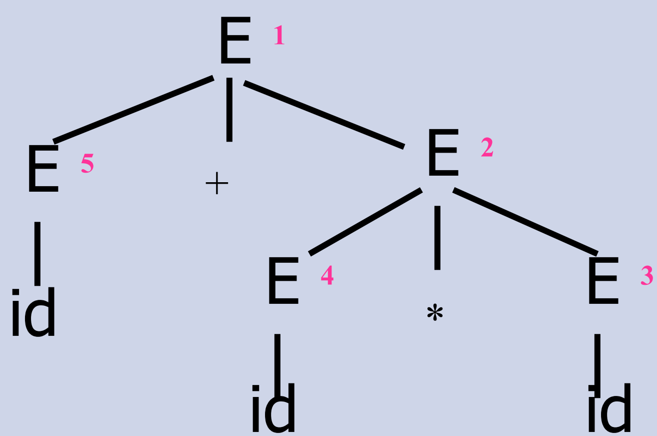


Parse Trees and Derivations



Preorder numbering

$$
\begin{aligned}
E &\Rightarrow E + E && (1)\\
&\Rightarrow id + E && (2)\\
&\Rightarrow id + E * E && (3)\\
&\Rightarrow id + id * E && (4)\\
&\Rightarrow id + id * id && (5)
\end{aligned}
$$



Reverse of postorder numbering

$$
\begin{aligned}
E &\Rightarrow E + E && (1)\\
&\Rightarrow E + E * E && (2)\\
&\Rightarrow E + E * id && (3)\\
&\Rightarrow E + id * id && (4)\\
&\Rightarrow id + id * id && (5)
\end{aligned}
$$

Grammar: Example

List of parameters in:

- Function definition
  - function sub(a,b,c)
- Function call
  - sub(a,1,2)

<argList>
⇒ **id ,** <arglist>
⇒ **id , id ,** <arglist>

<argList>
⇒ <arglist> **, id**
⇒<arglist> **, id, id**
⇒ … ⇒ **id (, id)***

<Fdef> → **function id (** <argList> **)**
<argList> → **id ,** <arglist> | **id**
<Fcall> → **id (** <parList> **)**
<parList> → <par> **,**<parlist>| <par>
<par> → **id** | **const**

<Fdef> → **function id (** <argList> **)**
<argList> → <arglist> **, id** | **id**
<Fcall> → **id (** <parList> **)**
<parList> → <parlist> **,**<par>| <par>
<par> → **id** | **const**

Grammar: Example

List of parameters

- If zero parameter is allowed, then ?

<Fdef> → **function id (** <argList> **)**|
function id ( )

<argList> → **id ,** <arglist> | **id**

<Fcall> → **id (** <parList> **) | id ( )**

<parList> → <par> **,**<parlist>| <par>

<par> → **id** | **const**

Work ?

<Fdef> → **function id (** <argList> **)**

<argList> → **id ,** <arglist> | **id** | λ

<Fcall> → **id (** <parList> **)**

<parList> → <par> **,**<parlist>| <par>

<par> → **id** | **const**

NO!
Generate
id , id , id ,

Grammar: Example

List of parameters

- If zero parameter is allowed, then ?

<Fdef> → **function id (** <argList> **)**|
function id ( )

<argList> → **id ,** <arglist> | **id**

<Fcall> → **id (** <parList> **) | id ( )**

<parList> → <par> **,**<parlist>| <par>

<par> → **id** | **const**

Work ?

<Fdef> → **function id (** <argList> **)**

<argList> → **id ,** <arglist> | **id** | λ

<Fcall> → **id (** <parList> **)**

<parList> → <par> **,**<parlist>| <par>

<par> → **id** | **const**

NO!
Generate
id , id , id ,

ขั้นตอนการ generate ไปเรื่อยๆ จนแต่ละตัวเสร็จ คือ

Grammar: Example

List of statements:

- No statement
- One statement:
  - s;
- More than one statement:
  - s; s; s;
- A statement can be a block of statements.
  - {s; s; s;}

Is the following correct?

{ {s; {s; s;} s; {}} s; }

<St> ::= λ | **s;** | **s;** <St> | **{** <St> **}** <St>

<St>

⇒ **{** <St> **}** <St>

⇒ **{** <St> **}**

⇒ **{ {** <St> **}** <St>**}**

⇒ **{ {** <St> **} s;** <St>**}**

⇒ **{ {** <St> **} s; }**

⇒ **{ { s;** <St> **} s;}**

⇒ **{ { s; {** <St> **}** <St> **} s;}**

⇒ **{ { s; {** <St> **} s;** <St> **} s;}**

⇒ **{ { s; {** <St> **} s; {** <St> **}** <St> **} s;}**

⇒ **{ { s; {** <St> **} s; {** <St> **} } s;}**

⇒ **{ { s; {** <St> **} s; {} } s;}**

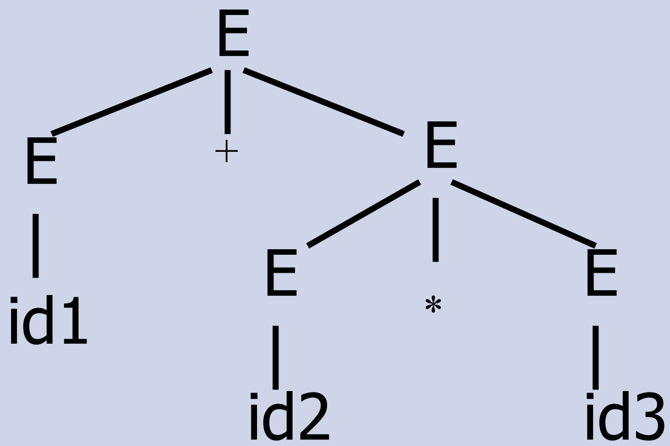
⇒ **{ { s; { s;** <St> **} s; {} } s;}**

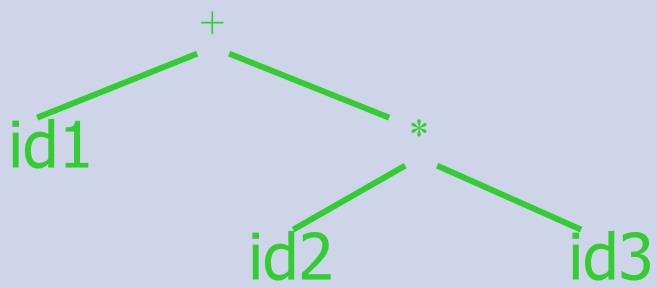
⇒ **{ { s; { s; s;} s; {} } s;}**

Abstract Syntax Tree

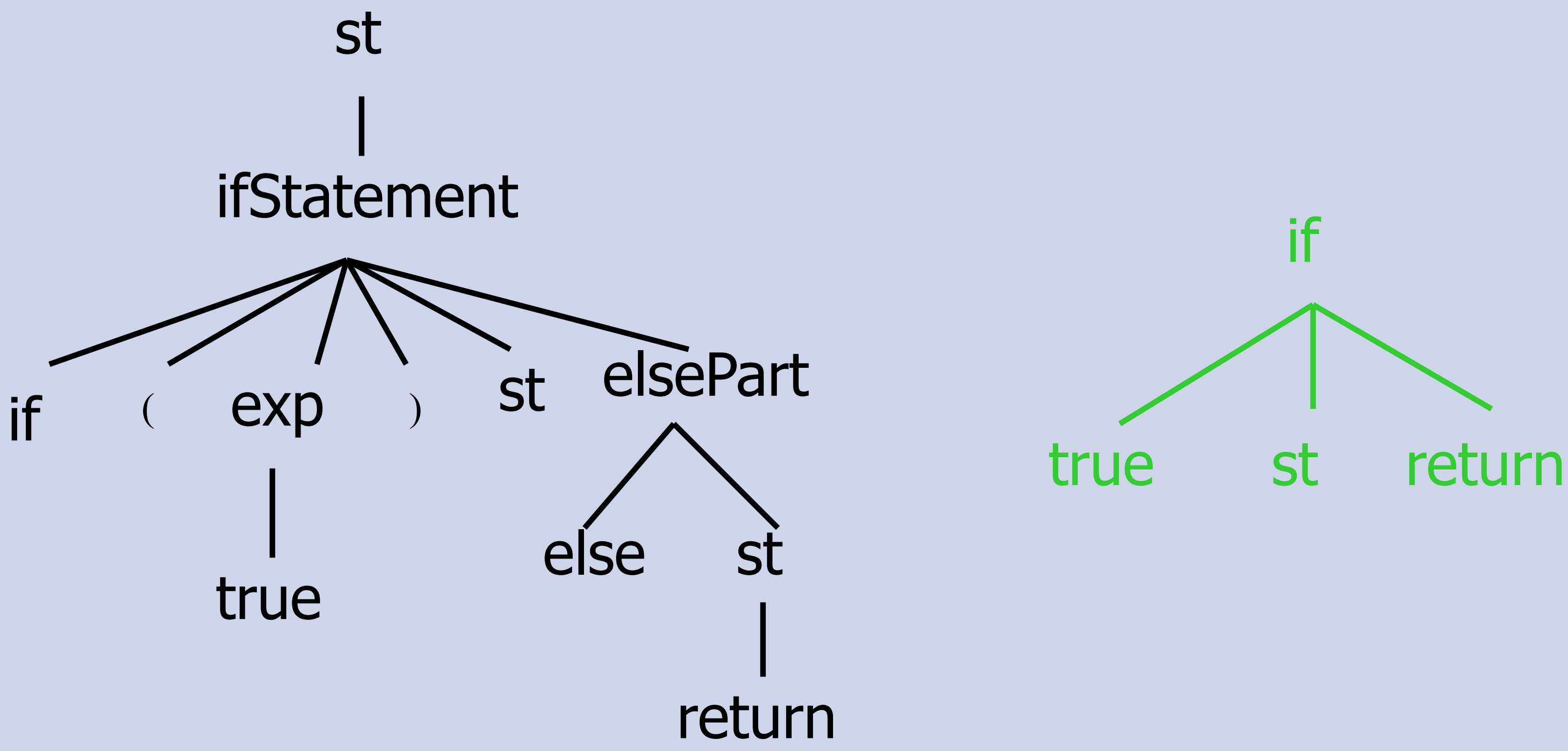
- Representation of actual source tokens
- Interior nodes represent operators.
- Leaf nodes represent operands.

Abstract Syntax Tree for Expression





Abstract Syntax Tree for If Statement



Ambiguous Grammar

- A grammar is ambiguous if it can generate two different parse trees for one string.
- Ambiguous grammars can cause inconsistency in parsing.

Example: Ambiguous Grammar

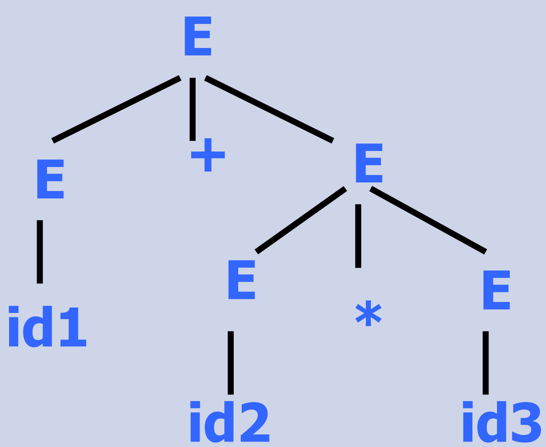
$E \rightarrow E + E$

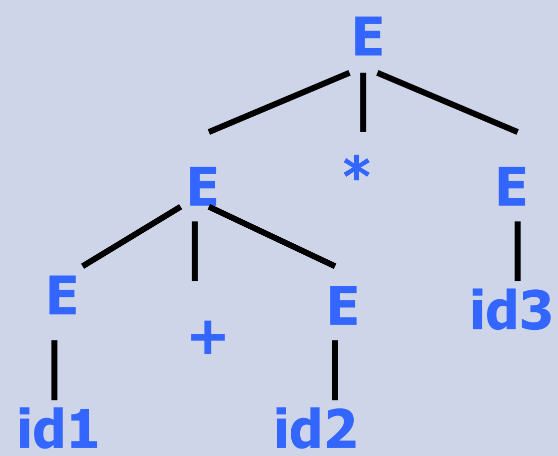
$E \rightarrow E - E$

$E \rightarrow E * E$

$E \rightarrow E / E$

$E \rightarrow id$





Ambiguity in Expressions

- Which operation is to be done first?
  - solved by precedence
    - An operator with higher precedence is done before one with lower precedence.
    - An operator with higher precedence is placed in a rule (logically) further from the start symbol.
  - solved by associativity
    - If an operator is right-associative (or left-associative), an operand in between 2 operators is associated to the operator to the right (left).
    - Right-associated : W + (X + (Y + Z))
    - Left-associated : ((W + X) + Y) + Z

Precedence

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow E * E$

$E \rightarrow E / E$

$E \rightarrow id$

$E \rightarrow E + E$

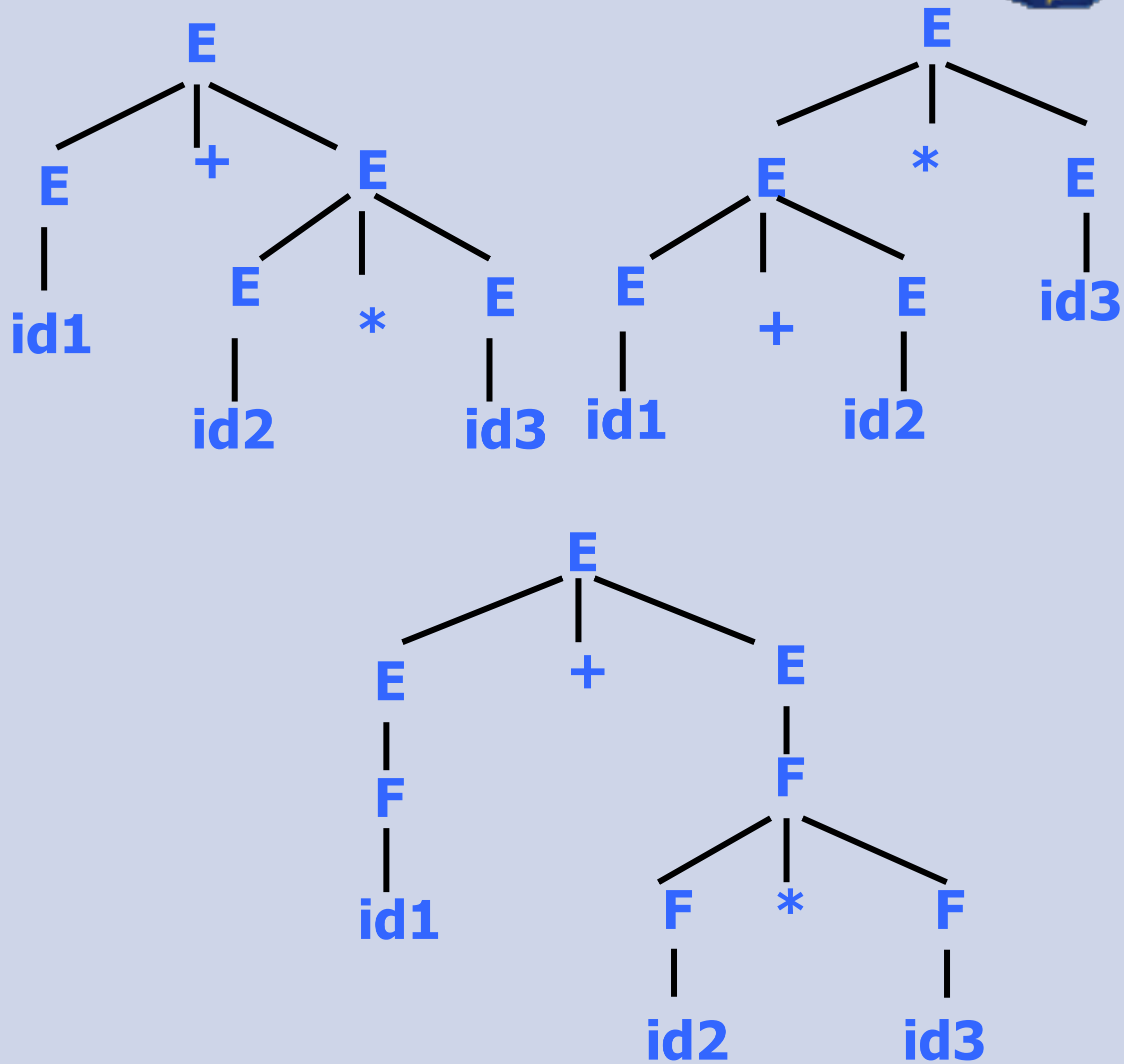
$E \rightarrow E - E$

$E \rightarrow F$

$F \rightarrow F * F$

$F \rightarrow F / F$

$F \rightarrow id$



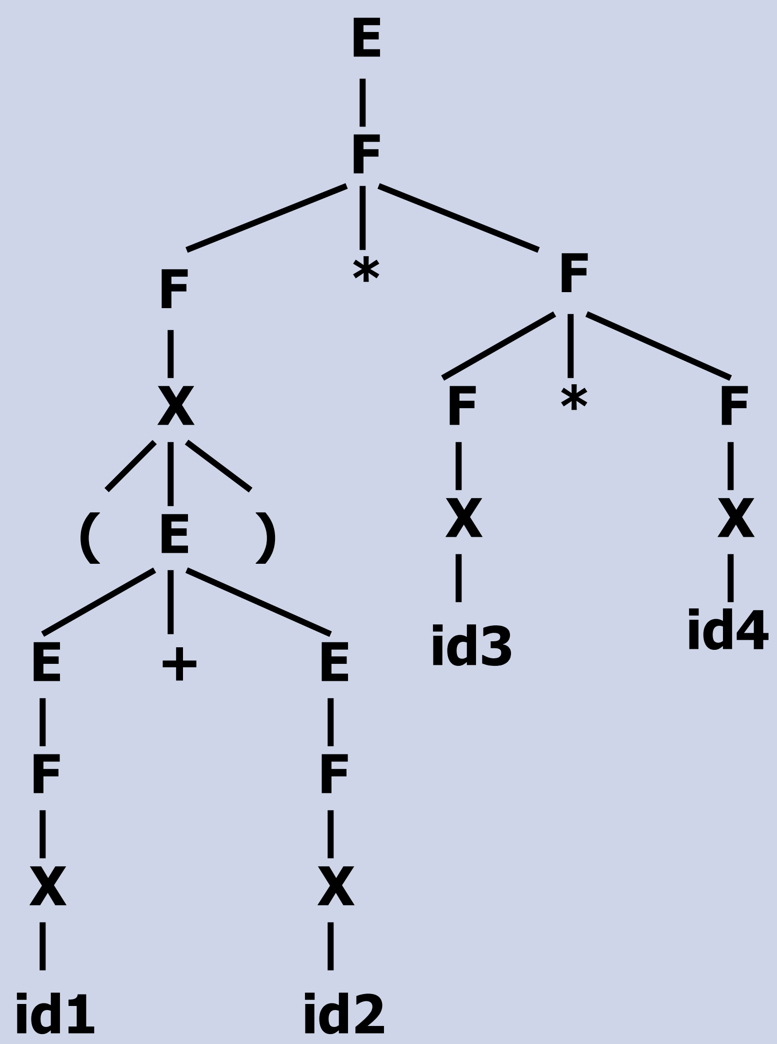
Precedence (cont'd)

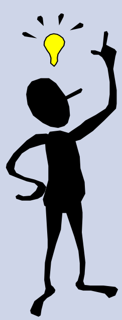
$E \rightarrow E + E \mid E - E \mid F$

$F \rightarrow F * F \mid F / F \mid X$

$X \rightarrow (E) \mid id$

(id1 + id2) * id3 * id4





Associativity

- Left-associative operators

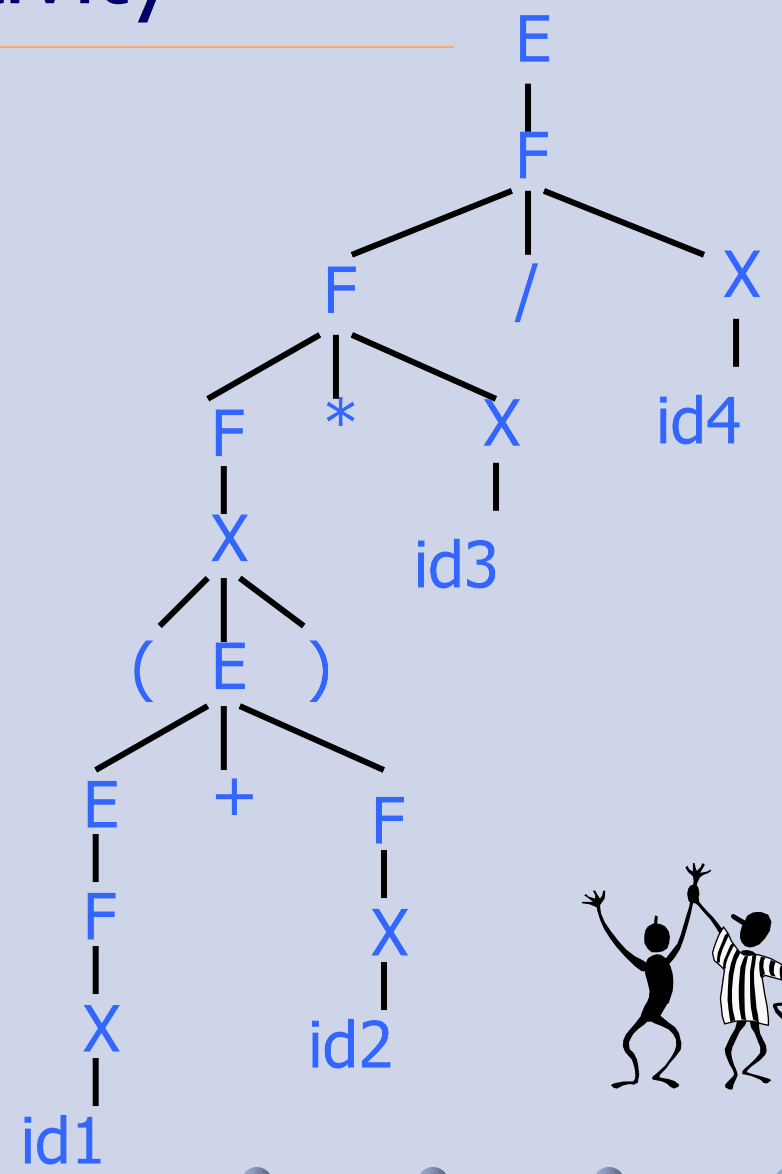
$E \rightarrow E + F \mid E - F \mid F$

$F \rightarrow F * X \mid F / X \mid X$

$X \rightarrow (E) \mid id$

(id1 + id2) * id3 / id4

= (((id1 + id2) * id3) / id4)



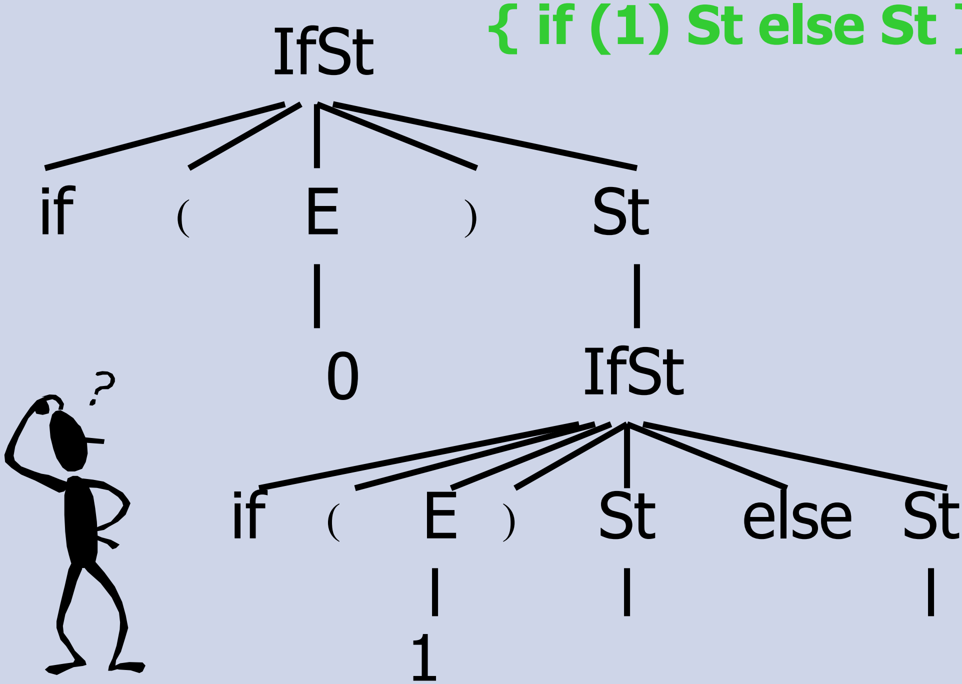
Ambiguity in Dangling Else

St $\rightarrow$ IfSt | ...

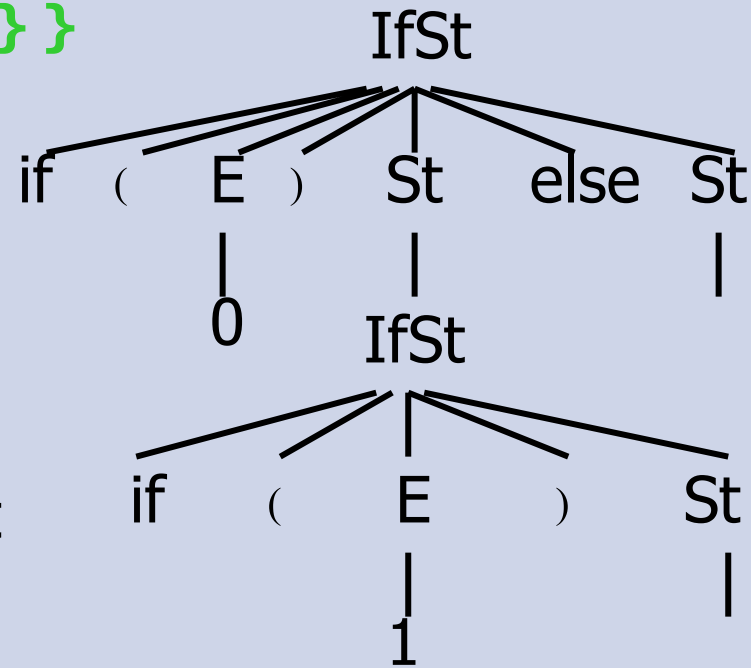
IfSt $\rightarrow$ if (E) St | if (E) St else St

E $\rightarrow$ 0 | 1 | ...

{ if (0)
{ if (1) St else St } }



{ if (0)
{ if (1) St }
else St }



Disambiguating Rules for Dangling Else

St →
  MatchedSt | UnmatchedSt

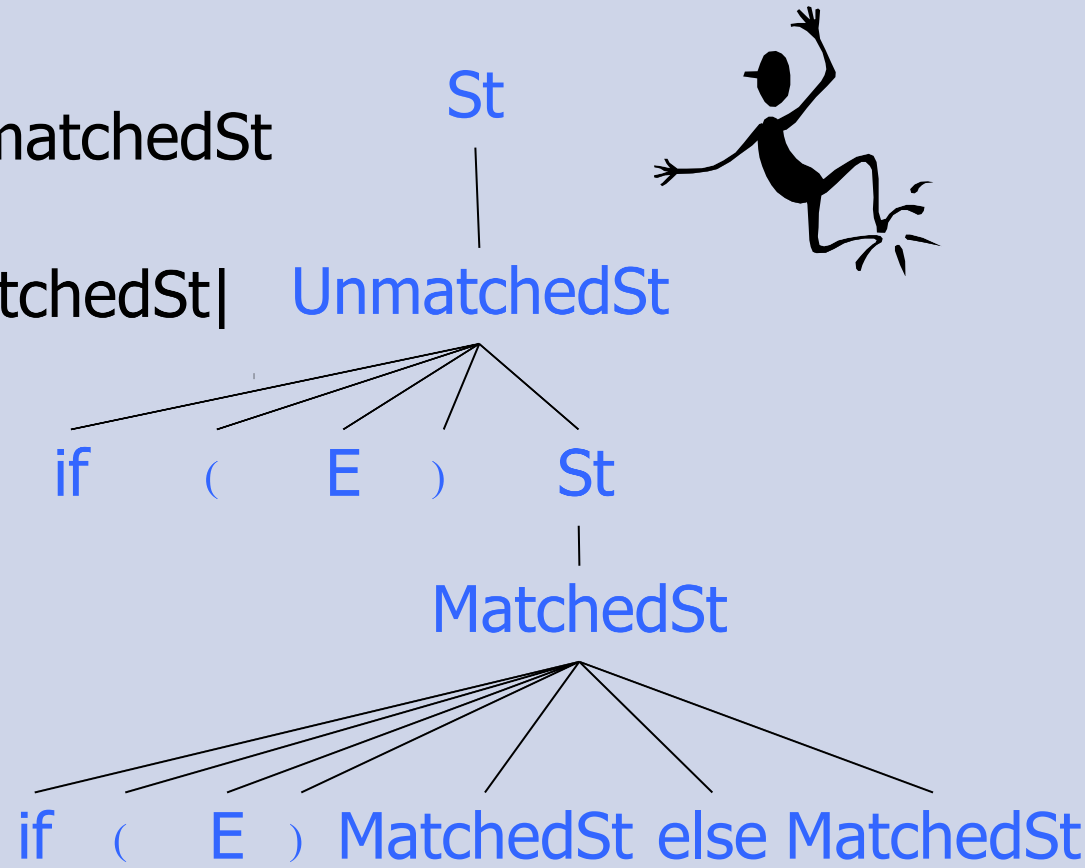
UnmatchedSt →
  if (E) St |
  if (E) MatchedSt else UnmatchedSt

MatchedSt →
  if (E) MatchedSt else MatchedSt|
  ...

E →
  0 | 1

- if (0) if (1) St else St

= if (0)
    if (1) St else St

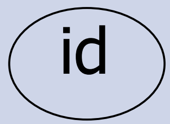


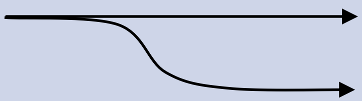
Extended Backus-Naur Form (EBNF)

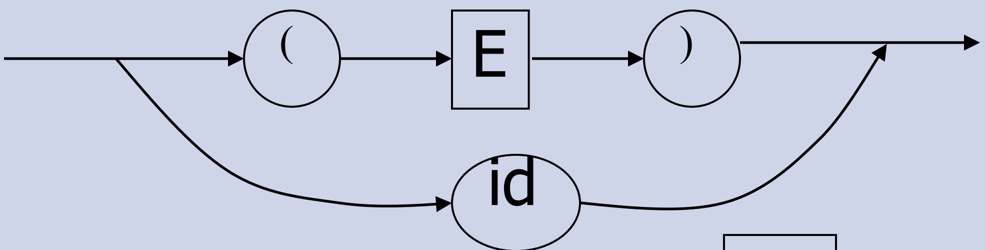
- Kleene's Star/ Kleene's Closure
  - Seq ::= St {; St}
  - Seq ::= {St ;} St
- Optional Part
  - IfSt ::= if (E) St [else St]
  - E ::= F [+ E] | F [- E]

Syntax Diagrams

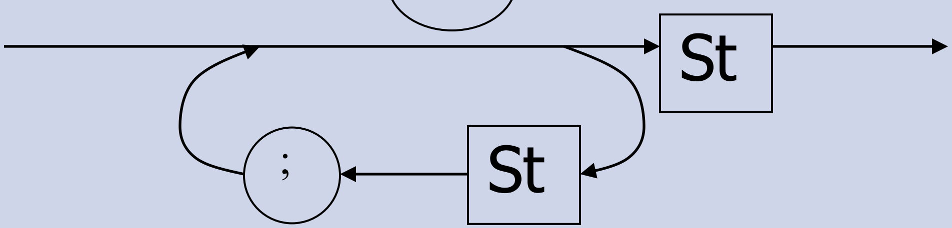
- Graphical representation of EBNF rules
  - nonterminals: 

  - terminals: 

  - sequences and choices: 
- Examples
  - X ::= (E) | id

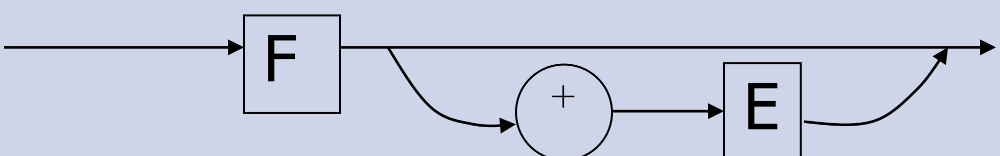


  - Seq ::= {St ;} St



optional

  - E ::= F [+ E]



Reading Assignment

- Louden, Compiler construction
  - Chapter 3