Compiler Principle

Prof. Dongming LU

Mar. 4th, 2024

Content

3 Parsing

Introduction

- **Lexical Analysis**: Create sequence of tokens from characters
- **Parsing**: Create abstract syntax tree from sequence of tokens

Introduction

Syntax: the way in which words are put together to form phrases, clauses, or sentences.

- **Need** more expressive power than **regular expression**

Context-free grammar: recursive power

Introduction

Parsing with CFGs

- Context-free grammars are (often) given by BNF expressions (Backus-Naur Form)
- More powerful than regular expressions
- CFGs are good for describing the overall syntactic structure of programs.

3.1 Context-free Grammars

Definition for CFG

Context-free grammars consist of:

- **Set of symbols:**
  - ✓ **Terminals** that denotes token types
  - ✓ **Non-terminals** that denotes a set of strings
- **Start symbol**
- **Rules:** **symbol → symbol symbol … symbol**
  - ✓ **Left-hand side: non-terminal**
  - ✓ **Right-hand side: terminals and/or non-terminals**
  - ✓ **Rules explain how to rewrite non-terminals (beginning with start symbol) into terminals**

An example of a CFG

- **Non-terminals:** S, E, L
- **Terminals:** id, nm, print, +, :=, (,), ;
- Rules:

1. S → S; S
2. S → id := E
3. S → print (L)

4. E → id
5. E → num
6. E → E + E
7. E → (S , E)

8. L → E
9. L → L , E

- **One sentence**:

  id := num; id := id + (id := num + num, id)

- **Source text:**

  a:= 7;

  b:=c +(d:5+6,d)

Derivations

A string is in the language of the CFG if and only if it is possible to derive the string using the following non-deterministic procedure.

1. **Begin with the start symbol**
2. **While any non-terminals exist, pick a non-terminal and rewrite it using a rule**
3. **Stop when all you have left are terminals**

Examples of Derivations

- **non-terminals:** S, E, L
- **terminals:** id, nm, print, +, :=, (,), ;
- **rules:**

1. S $\rightarrow$ S; S
2. S $\rightarrow$ id := E
3. S $\rightarrow$ print (L)
4. E $\rightarrow$ id
5. E $\rightarrow$ num
6. E $\rightarrow$ E + E
7. E $\rightarrow$ (S , E)
8. L $\rightarrow$ E
9. L $\rightarrow$ L , E

S

Derive me!

id := num ; print (num)

Examples of Derivations

non-terminals: S, E, L

terminals: id, nm, print, +, :=, (,), ;

rules:

1. S → S; S
2. S → id := E
3. S → print (L)
4. E → id
5. E → num
6. E → E + E
7. E → (S , E)
8. L → E
9. L → L , E

Derive me!

S

id := E

id := num ; print (num)

Examples of Derivations

non-terminals: S, E, L

terminals: id, nm, print, +, :=, (,), ;

rules:

1. $S \rightarrow S; S$
2. $S \rightarrow id := E$
3. $S \rightarrow print (L)$
4. $E \rightarrow id$
5. $E \rightarrow num$
6. $E \rightarrow E + E$
7. $E \rightarrow (S , E)$
8. $L \rightarrow E$
9. $L \rightarrow L , E$

Derive me!

S

id := E

can't make progress

$\overbrace{\text{id := num ; print (num)}}^{\cancel{E}}$

Examples of Derivations

non-terminals: S, E, L

terminals: id, nm, print, +, :=, (,), ;

rules:

1. **S → S; S**
2. **S → id := E**
3. **S → print (L)**
4. **E → id**
5. **E → num**
6. **E → E + E**
7. **E → (S , E)**
8. **L → E**
9. **L → L , E**

S

Derive me!

id := num ; print (num)

Examples of Derivations

non-terminals: S, E, L

terminals: id, nm, print, +, :=, (,), ;

rules:

1. S → S; S
2. S → id := E
3. S → print (L)
4. E → id
5. E → num
6. E → E + E
7. E → (S , E)
8. L → E
9. L → L , E

Derive me!

S
S ; S

id := num ; print (num)

Examples of Derivations

non-terminals: S, E, L

terminals: id, nm, print, +, :=, (,), ;

rules:

1. **S → S; S**
2. **S → id := E**
3. **S → print (L)**
4. **E → id**
5. **E → num**
6. **E → E + E**
7. **E → (S , E)**
8. **L → E**
9. **L → L , E**

Derive me!

S
S ; S
id := E ; S

id := num ; print (num)

Examples of Derivations

non-terminals: S, E, L

terminals: id, nm, print, +, :=, (,), ;

rules:

1. S → S; S
2. S → id := E
3. S → print (L)
4. E → id
5. E → num
6. E → E + E
7. E → (S , E)
8. L → E
9. L → L , E

Derive me!

S
S ; S
id := E ; S
id := num ; S
id := num ; print (L)
id := num ; print (E)
id := num ; print (num)

Examples of Derivations

rules:

1. $S \rightarrow S; S$
2. $S \rightarrow id := E$
3. $S \rightarrow print (L)$
4. $E \rightarrow id$
5. $E \rightarrow num$
6. $E \rightarrow E + E$
7. $E \rightarrow (S , E)$
8. $L \rightarrow E$
9. $L \rightarrow L , E$

S
S ; S
id := E ; S
id := num ; S
id := num ; print (L)
id := num ; print (E)
id := num ; print (num)

left-most derivation

S
S ; S
S ; print (L)
S ; print (E)
S ; print (num)
id := E; print (num)
id := num ; print (num)

right-most derivation

Parse Trees

Example:

S

S ; S

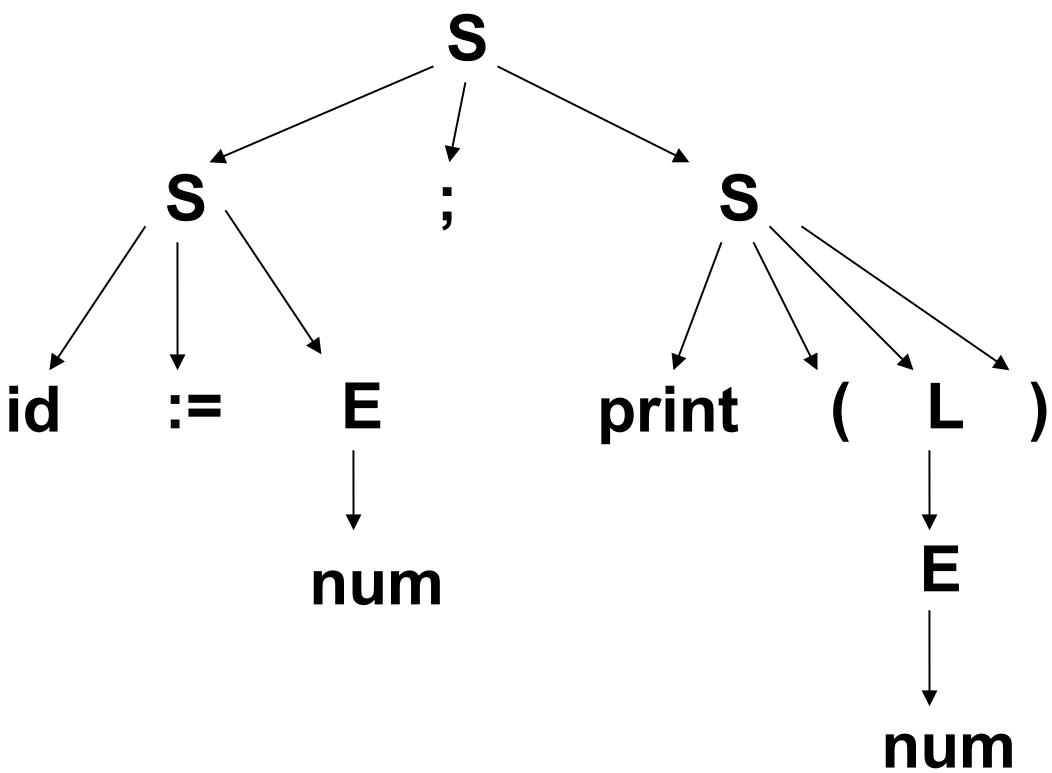
id := E ; S

id := num ; S

id := num ; print (L)

id := num ; print (E)

id := num ; print (num)



Parse Trees

Representing derivations as a tree

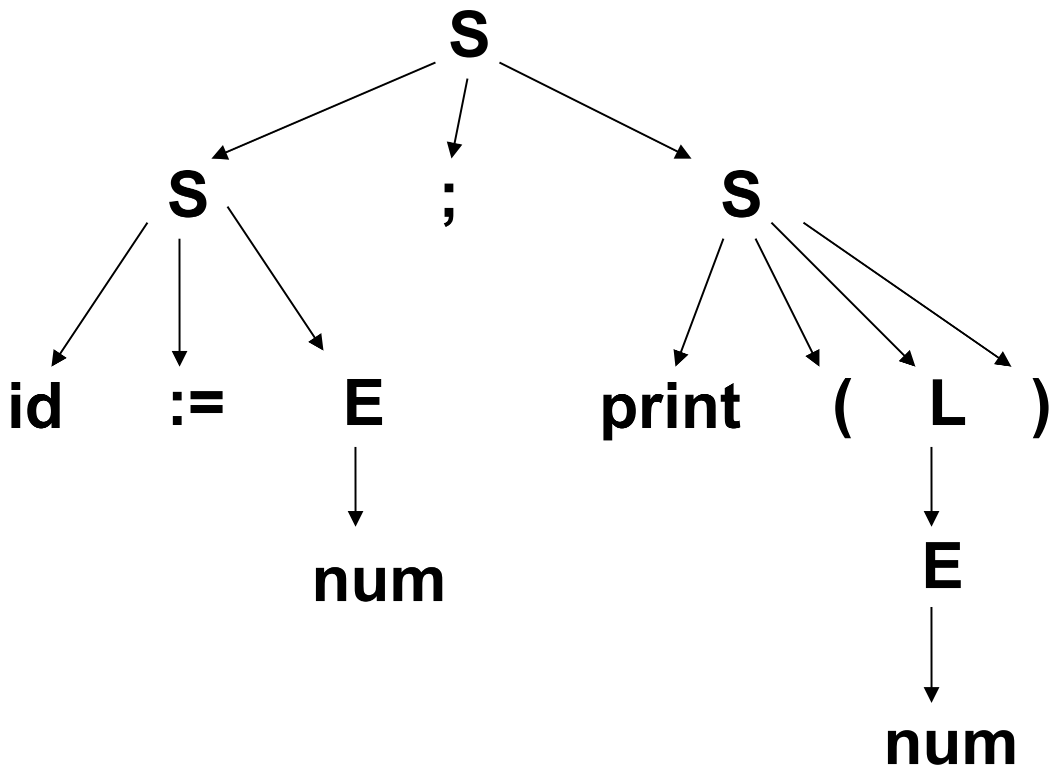
- Each **internal node** is labeled with **a non-terminal**
- Each **leaf node** is labeled with **a terminal**
- Each use of **a rule** in **a derivation** explains how to generate children in the parse tree from the parents

Parse Trees

Example: 2 different derivations, but 1 same tree

S
S ; S
id := E ; S
id := num ; S
id := num ; print (L)
id := num ; print (E)
id := num ; print (num)

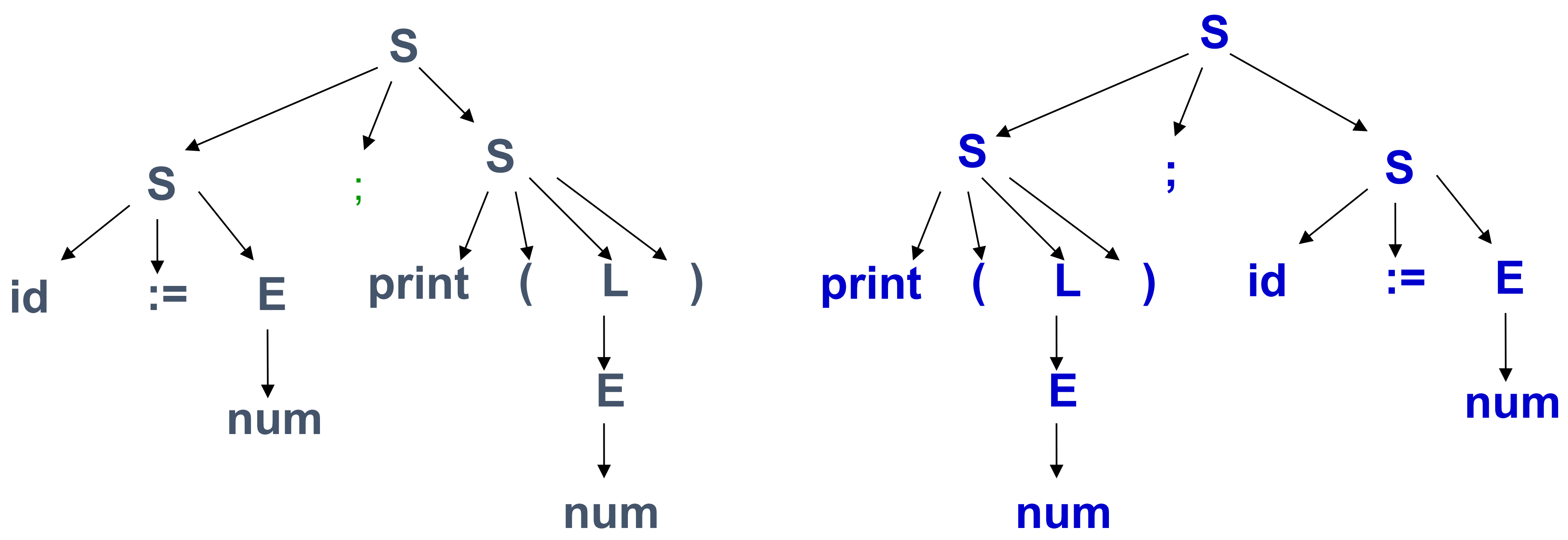
S
S ; S
S ; print (L)
S ; print (E)
S ; print (num)
id := E; print (num)
id := num ; print (num)



Parse Trees

Parse trees have meaning

- **Order of children, nesting of subtrees is significant**



Ambiguous Grammars

A grammar is ambiguous if the same sequence of tokens can give rise to two or more parse trees.

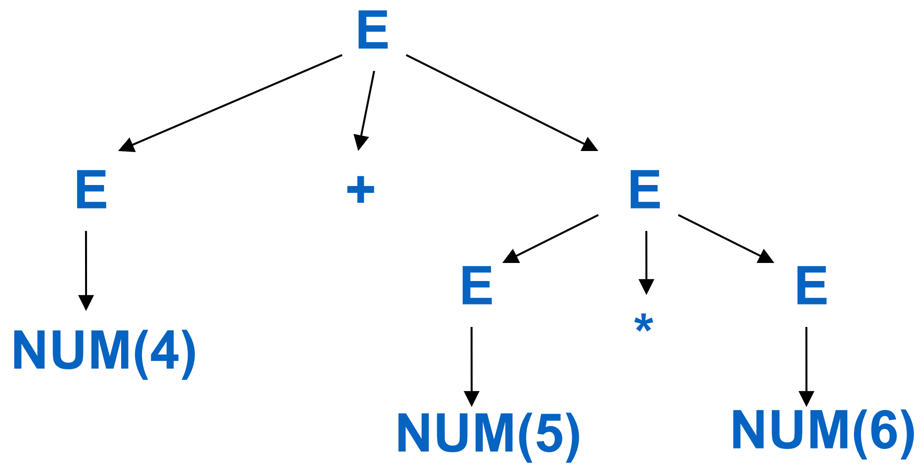
Ambiguous Grammars

non-terminals:
  E

terminals:
  ID
  NUM
  +
  *

E → ID
  | NUM
  | E + E
  | E * E

The sequence of characters: 4 + 5 * 6



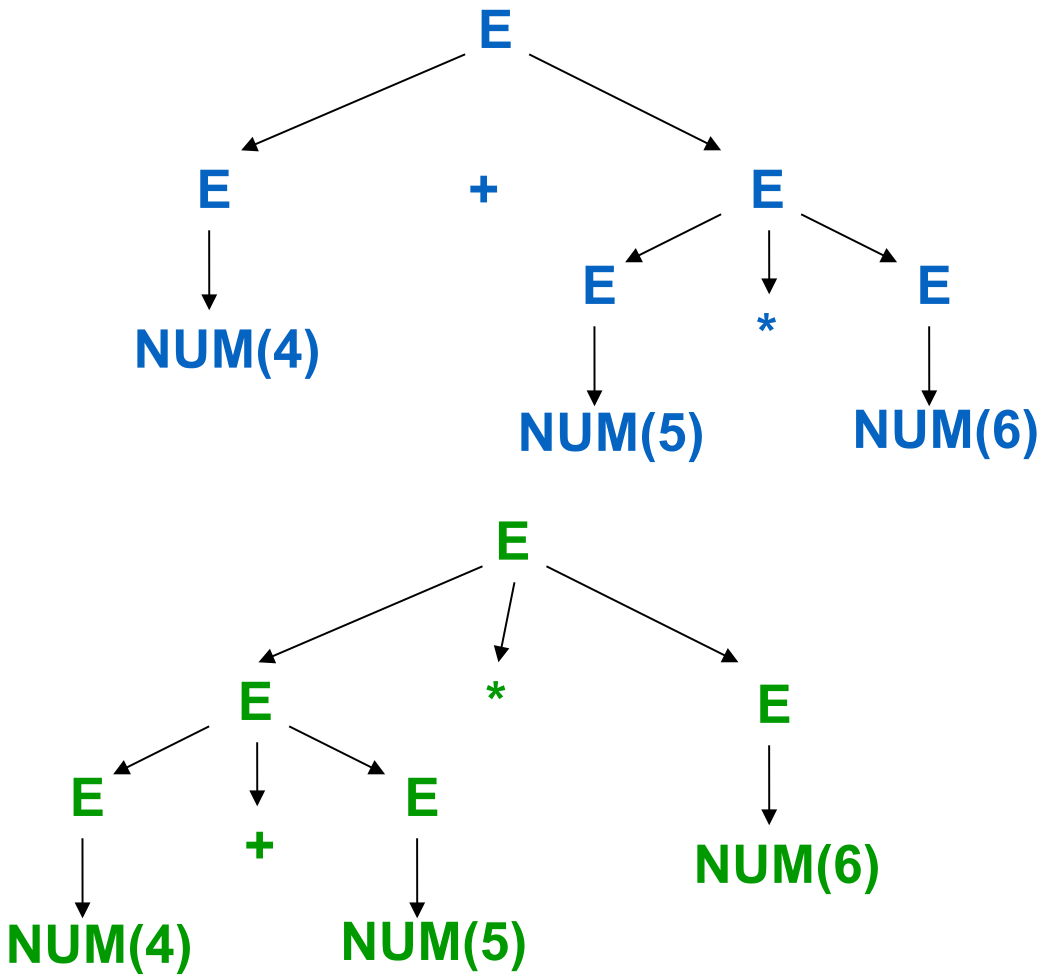
Ambiguous Grammars

non-terminals:
E

terminals:
ID
NUM
+
*

$E \rightarrow$ ID
| NUM
| E + E
| E * E

The sequence of characters: 4 + 5 * 6



Ambiguous Grammars

$E \rightarrow E + T$
$E \rightarrow E - T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow T / F$
$T \rightarrow F$
$F \rightarrow \text{id}$
$F \rightarrow \text{num}$
$F \rightarrow (E)$

- This grammar accepts the same set of sentences as the ambiguous grammar.
- Eliminating ambiguity by transforming the grammar.

There are some languages (sets of strings) that have ambiguous grammars but no unambiguous grammar, such languages may be problematic

End-Of-File Marker

- Use **$** to represent end of file
- Suppose ***S*** is the start symbol of a grammar.
- To indicate that $ must come after a complete *S*-phrase
- Augment the grammar with a new start symbol *S′* and a new production ***S′* → *S*$**.

$S \longrightarrow E\ \$$
$E \longrightarrow E + T$
$E \longrightarrow E - T$
$E \longrightarrow T$
$T \longrightarrow T * F$
$T \longrightarrow T\ /\ F$
$T \longrightarrow F$
$F \longrightarrow \text{id}$
$F \longrightarrow \text{num}$
$F \longrightarrow (E)$

The end of Chapter 3(1)