Compilation Principle
编 译 原 理

第5讲：语法分析(2)

张献伟

xianweiz.github.io

DCS290, 3/8/2022

Review Example

- $vim test.c

```
void main() {
 int;
 int a,;
 int b, c;
}
```

- $clang -cc1 -dump-tokens ./test.c
- $clang -o test test.c

```
test.c:1:1: warning: return type of 'main' is not 'int' [-Wmain-return
void main() {
^
test.c:1:1: note: change return type to 'int'
void main() {
^~~~
int
test.c:2:3: warning: declaration does not declare anything [-Wmissing-
ns]
  int;
  ^~~
test.c:3:9: error: expected identifier or '('
  int a,;
        ^
2 warnings and 1 error generated.
```

```
void 'void'
identifier 'main'
l_paren '('
r_paren ')'
l_brace '{'
int 'int'
semi ';'
int 'int'
identifier 'a'
comma ','
semi ';'
int 'int'
identifier 'b'
comma ','
identifier 'c'
semi ';'
r_brace '}'
eof ''
```

Syntax Analysis[语法分析]

- Informal description of variable declarations in C[变量声明]
  - Starts with *int* or *float* as the first token[类型]
  - Followed by one or more *identifier* tokens, separated by token *comma*[逗号分隔的标识符]
  - Followed by token *semicolon*[分号]
- To *check whether a program is well-formed* requires a specification of *what is a well-formed program*[语法定义]
  - The specification be precise[正确]
  - The specification be complete[完备]
    - Must cover all the syntactic details of the language
  - The specification must be convenient[便捷] to use by both language designer and the implementer
- A **context free grammar** meets these requirements

Context Free Grammar[上下文无关文法]

- Formal definition[形式化定义]: 4 components **{T, N, s, δ}**
  - *T* is a finite set of terminals
  - *N* is a finite set of non-terminals
  - *S* is a special nonterminal (from N) called the start symbol
  - δ is a finite set of production rules of the form such as A→ α, where A is from N and α from (N ∪ T) ∗
- CFG of variable declarations
  - $\{\underbrace{\{\mathrm{id}\ ,\ \mathrm{int}\ \mathrm{float}\ ;\}}_{\mathbf{T}},\ \underbrace{\{\mathit{declaration}\ \mathit{type}\ \mathit{idlist}\}}_{\mathbf{N}},\ \underbrace{\mathit{declaration}}_{\mathbf{s}},\ \delta\}$
- Production rules (δ)

  declaration → *type idlist ;*

  idlist → id | *idlist* , id

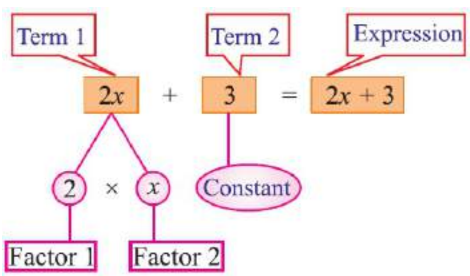
  type → int | float

Notational Conventions[标识规范]

- These symbols are terminals[终结符]
  - Lowercase letters early in the alphabet, e.g., *a*, *b*, *c*[靠前小写字母]
  - Operator symbols such as +, *, ...[运算符]
  - Punctuation symbols such as (, , ...[标点符号]
  - Digits 0, 1, ..., 9[数字]
  - Boldface strings such as **id** or **if**, each is a single terminal symbol
- These symbols are non-terminals[非终结符]
  - Uppercase letters early in alphabet, e.g., A, B, C[靠前大写字母]
  - The letter S, which, when it appears, is usually the start symbol
  - Lowercase, italic names such as *expr* or *stmt*[小写斜体]
  - When discussing programming constructs, uppercase letters may represent non-terminals for the constructs
    - E.g., *E*: expression[表达式], *T*: term[项], *F*: factor[因子]

Notational Conventions (cont.)

- Uppercase letters late in alphabet, e.g., X, Y, Z, represent grammar symbols
  - Either non-terminals or terminals
- Lowercase letters late in alphabet, chiefly u, v, ..., z, represent (possibly empty) strings of terminals
- Lowercase Greek letters, e.g.,α, β, γ represent (possibly empty) strings of grammar symbols
  - $A \rightarrow \alpha$
- Unless stated otherwise, the head of the first production is the start symbol [开始符号]



$E \rightarrow E + T \mid E - T \mid T$

$T \rightarrow T * F \mid T / F \mid F$

$F \rightarrow (E) \mid$ **id**

Start symbol: E

Nonterminals: E, T and F

Terminals: everything else

Production Rule and Derivation[推导]

- **Production rule**[产生规则]: *LHS* → *RHS*
  - Aliases[别名]: *LHS* ≡ head, *RHS* ≡ body
  - Meaning[含义]: *LHS* can be constructed (or replaced) with *RHS*
- **Derivation**[推导]: a series of applications of production rules
  - Replace a non-terminal by the corresponding *RHS* of a production
- $\beta \Rightarrow \alpha$
  - Meaning: string α is derived from β
  - $\beta \Rightarrow \alpha$: derives in one step
  - $\beta \Rightarrow^* \alpha$: derives in zero or more steps
  - $\beta \Rightarrow^+ \alpha$: derives in one or more steps
- Example: $A \Rightarrow 0A \Rightarrow 00B \Rightarrow 000$
  - $A \Rightarrow^* 000$
  - $A \Rightarrow^+ 000$

Derivation[推导]

- If $S \Rightarrow^* \alpha$, where S is the start symbol of grammar G

- α: **sentential form** of G[句型]
  - A sentential form may contain <u>both terminals and non-terminals</u> (and can be empty)

S = subject, V = verb, O = object
SV: She laughed.
SVO: She opened the door.

- α: **sentence** of G[句子]
  - A sentential form with <u>no non-terminals</u>

- **Language**[语言] generated by a grammar
  - $L(G) = \{w: S \Rightarrow {}^*w,\ w \in V_T^*\}$
  - A string of terminal *w* is in L(G), **iff** *w* is a sentence of G (or $S \Rightarrow^* w$)

Example

- Grammar $G = \{T, N, s, \delta\}$
  - $T = \{0, 1\}$
  - $N = \{A, B\}$
  - $s = A$
  - $\delta = \{ A \rightarrow 0A \mid 1A \mid 0B, B \rightarrow 0 \}$

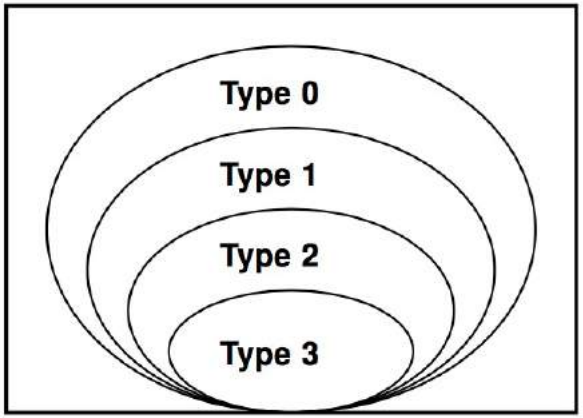
- Derivation: from grammar to language[文法到语言]
  - $A \Rightarrow 0A \Rightarrow 00B \Rightarrow 000$
  - $A \Rightarrow 1A \Rightarrow 10B \Rightarrow 100$
  - $A \Rightarrow 0A \Rightarrow 00A \Rightarrow 000B \Rightarrow 0000$
  - $A \Rightarrow 0A \Rightarrow 01A \Rightarrow \ldots$
  - … …

Sentence

Sentential form

Language Classification: Chomsky

- **Language classification** based on form of grammar rules
- Four types of grammars:
  - Type 0 — unrestricted grammar
    - 0型文法 – 无限制文法
  - Type 1 — context sensitive grammar(CSG)
    - 1型文法 – 上下文有关文法
  - Type 2 — context free grammar (CFG)
    - 2型文法 – 上下午无关文法
  - Type 3 — regular grammar
    - 3型文法 – 正则文法



- Regular Grammar ⊆ CFG ⊆ CSG ⊆ Unrestricted Grammar

Chomsky hierarchy

In 1957, Noam Chomsky published *Syntactic Structures*, an landmark book that defined the so-called Chomsky hierarchy of languages

Type 0: Unrestricted Grammar

- Form of rules $\alpha \to \beta$
  - where $\alpha \in (N \cup T)^+$, $\beta \in (N \cup T)^*$
- Implied restrictions
  - LHS: no ε allowed
- Example:
  - aB → aCD: LHS is shorter than RHS
  - aAB → aB : LHS is longer than RHS
  - A → ε: ε-productions are allowed
- Derivations
  - Derivation strings may contract and expand repeatedly (since LHS may be longer or shorter than RHS)
  - Unbounded number of productions before target string

Type 1: Context Sensitive Grammar

- Form of rules: $\alpha A\beta \rightarrow \alpha\gamma\beta$
  - where $A \in N$, $\alpha, \beta \in (N \cup T)^*$, $\gamma \in (N \cup T)^+$
- Replace A by γ only if found <u>in the context of α and β</u>
- Implied restrictions
  - LHS: shorter or equal to RHS
  - RHS: no ε allowed
- Example:
  - aAB→aCB: replace A with C when in between a and B
  - A → C: replace A with C regardless of context
- Derivations
  - Derivation strings may only expand
  - Bounded number of derivations before target string

Σ

Type 2: Context Free Grammar

- Form of rules: $A \rightarrow \gamma$
  - where $A \in N$, $\gamma \in (N \cup T)^+$
- Replace A by γ (no context can be specified)
- Implied restrictions
  - LHS: a single non-terminal
  - RHS: no ε allowed
    - Sometimes relaxed to simplify grammar but rules can always be rewritten to exclude ε-productions
- Example:
  - A → aBc: replace A with aBc regardless of context

L = $\{ a^n b^n \mid n \geq 0 \}$ is **NOT regular** but **IS** a **context-free language**.

For the following CFG G = $< T, N, S, \delta >$ generates L:
$T = \{$ a, b $\}$, $N = \{$ S $\}$ and $\delta = \{$ S -> aSb , S -> ab $\}$

Type 3: Regular Grammar

- Form of rules $A \to \alpha$,or $A \to \alpha B$
  - where $A,B \in N$, $\alpha \in T$
- In terms of FA:
  - Move from state A to state B on input α
- Implied restrictions
  - LHS: a single non-terminal
  - RHS: a terminal or a terminal followed by a non-terminal
- Example: $A \to 1A \mid 0$
  - RE: **1*0**
- Derivation:
  - Derivation string length increases by 1 at each step

SUN YAT-SEN UNIVERSITY 中山大學

In Practice[实际中]

- Every regular language is a context-free language
  - Context-free languages more general than regular languages
- If PLs are context-sensitive, why use CFGs for parsing?
  - Perfectly suited to describing recursive syntax of expressions and statements
  - CSG parsers are provably inefficient
  - Most PL constructs are context-free
    - if-stmt, declarations
  - The remaining context-sensitive constructs can be analyzed at the semantic analysis stage
    - e.g. def-before-use, matching formal/actual parameters
- In PLs
  - Regular language for lexical analysis
  - Context-free language for syntax analysis

Grammar and Derivation[文法与推导]

- **Grammar** is used to derive string or construct parser[文法]
- A **derivation** is a sequence of applications of rules[推导]
  - Starting from the start symbol
  - S⇒... ⇒... ⇒... ⇒ (sentence)
  - There are choices at each sentential form
    - choice of the nonterminal to be replaced
    - choice of a rule corresponding to the nonterminal
- Instead of choosing the nonterminal to be replaced, in an *arbitrary* fashion, it is possible to make an *uniform* choice at each step
- **Leftmost** and **Rightmost** derivations[最左和最右推导]
  - At each derivation step, leftmost derivation always replaces the leftmost non-terminal symbol
  - Rightmost derivation always replaces the rightmost one

Example

- Two derivations of string “id * id + id * id” using grammar:

  $E \rightarrow E*E \mid E+E \mid (E) \mid \mathrm{id}$

- Leftmost derivation[最左推导]
  - $E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow \mathrm{id} * E + E \Rightarrow \mathrm{id} * \mathrm{id} + E \Rightarrow \ldots \Rightarrow \mathrm{id} * \mathrm{id} + \mathrm{id} * \mathrm{id}$
- Rightmost derivation[最右推导]
  - $E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * \mathrm{id} \Rightarrow E + \mathrm{id} * \mathrm{id} \Rightarrow \ldots \Rightarrow \mathrm{id} * \mathrm{id} + \mathrm{id} * \mathrm{id}$

- Derivations can be summarized as a parse tree[分析树]