

Chapter 2 Overview of High-Level Programming Languages

第二章 高级语言及其语法描述

- 程序设计语言的定义
- 高级语言的一般特性
- 程序设计语言的语法描述

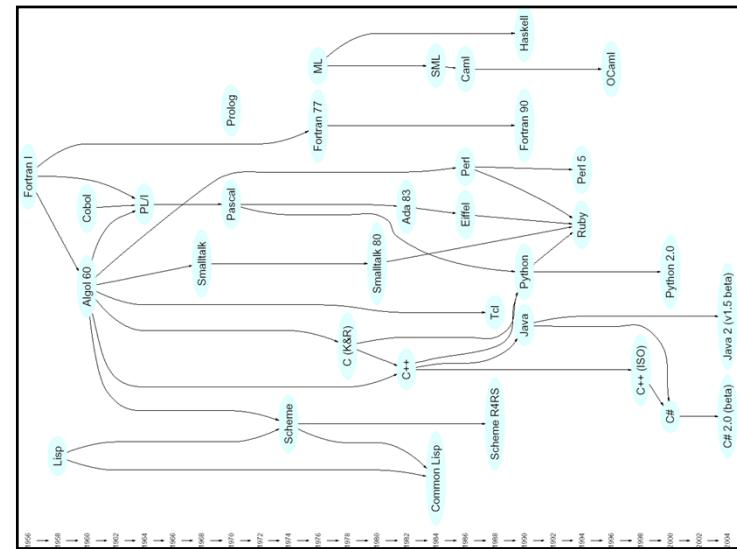
2

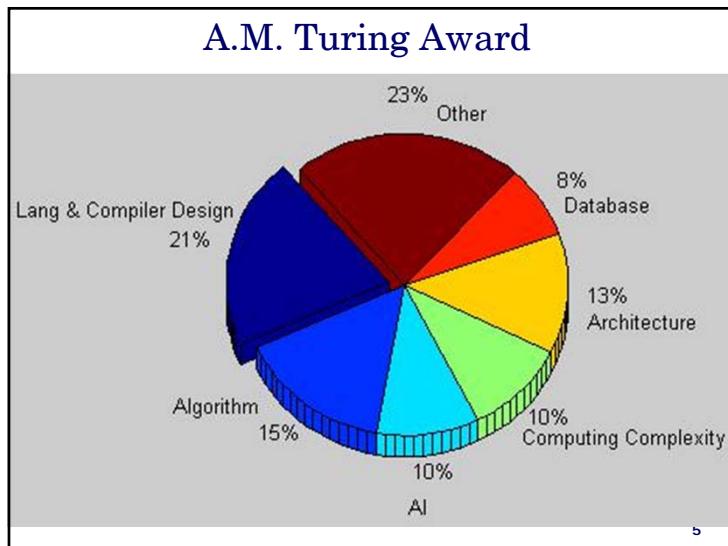
程序设计语言历史

- 50's: Fortran & Lisp
- 60's: Algol, PL/1, Simula67
- 70's: Pascal, Prolog, C, ML, Smalltalk
- 80's: Common Lisp, C++, Ada
- 90's: Fortran90, Ansi C, Tcl, Java
- 00's: C#, Java2

▲ As of May 2006 [The Encyclopedia of Computer Languages](#)
by Murdoch University, Australia lists 8512 computer
languages.

3





Year	Recipients	Citation
1966	Alan J. Perlis	For his influence in the area of advanced programming techniques and compiler construction
1971	John McCarthy	Lisp
1972	Edsger W. Dijkstra	ALGOL
1974	Donald E. Knuth	For his major contributions to the analysis of algorithms and the design of programming languages
1975	Allen Newell and Herbert A. Simon	..., and list processing
1976	Michael O. Rabin and Dana S. Scott	"Finite Automata and Their Decision Problem,"
1977	John Backus	FORTRAN
1978	Robert W. Floyd	the theory of parsing, the semantics of programming languages, automatic program verification, automatic program synthesis, and analysis of algorithms
1979	Kenneth E. Iverson	APL
1980	C. Antony R. Hoare	For his fundamental contributions to the definition and design of programming languages

1983	Ken Thompson& Dennis M. Ritchie	UNIX operating system
1984	Niklaus Wirth	For developing a sequence of innovative computer languages, EULER, ALGOL-W, MODULA and PASCAL
1987	John Cocke	For significant contributions in the design and theory of compilers, ... (RISC)
1991	Robin Milner	..., ML, CCS,
1995	Manuel Blum	computational complexity theory and program checking.
1996	Amir Pnueli	to program and systems verification.
2001	Ole-Johan Dahl, Kristen Nygaard	Simula I and Simula 67.
2003	Alan Kay	Smalltalk
2005	Peter Naur	ALGOL 60, to compiler design, and to the art and practice of computer programming.
2006	Frances E. Allen	optimizing compiler techniques that laid the foundation for modern optimizing compilers and automatic parallel execution.
2008	Barbara Liskov	foundations of programming language and system design, especially related to data abstraction, fault tolerance, and distributed computing.

2.1 程序设计语言的定义

- 语法和语义
- 类型系统

8

语法和语义

- 字母表 是符号的非空有穷集合。程序是字母表上的一个字符串
 - ⊕ {A, B, ..., Y, Z, 0, 1, ..., 9, 空格, +, -, *, /, =, <, >, (,), [,], {, }, ', \, ., :, ^}
- 词法单位定义 语言文法的一部分(词法规则)
 - ⊕ 常数、标识符、保留字、运算符、分界符、特殊符号
- 语法单位定义 语言文法的另一部分(语法规则)
 - ⊕ 表达式、语句、复合语句、程序等
- 语义描述 单词符号和语法单位的含义
 - ⊕ 函数副作用、类型推理规则、各类语句的执行规则等

9

Syntax

- Programming language syntax is usually defined using a combination of regular expressions (for lexical structure) and Backus-Naur Form (for grammatical structure).

```

<real-number> ::= <int-part> . <fraction>
<int-part> ::= <digit> | <int-part><digit>
<fraction> ::= <digit> | <digit><fraction>
<digit> ::= 0|1|2|3|4|5|6|7|8|9
  
```

10

Type system

- A type system defines how a programming language classifies values and expressions into types, how it can manipulate those types and how they interact. This generally includes a description of the data structures that can be constructed in the language.

11

Typed versus untyped languages

- A language is *typed* if the specification of every operation defines types of data to which the operation is applicable, with the implication that it is not applicable to other types.

12

Static versus dynamic typing

■ Static typing

- ⊕ In *static typing* all expressions have their types determined prior to the program being run (typically at compile-time).
- ⊕ *manifestly typed*: C++, C# and Java
- ⊕ *type-inferred*: Haskell and ML

■ Dynamic typing

- ⊕ determines the type-safety of operations at runtime; in other words, types are associated with runtime values rather than textual expressions
- ⊕ Ruby, Lisp, JavaScript, and Python

13

Weak and strong typing

■ **Weak typing** allows a value of one type to be treated as another, for example treating a string as a number

■ **Strong typing** prevents the above. An attempt to perform an operation on the wrong type of value raises an error.

14

例 1

```
#include <stdio.h>

int main(void)
{
    char s[] = "Test String";
    float x = 0.0;
    int y = 0;

    printf("Hello World\n");
    return 0;
}
```

15

例 2

```
> (defun fact(n)
  (if (< n 1) 1
      (* n (fact (1- n)))))

fact
> (fact 6)
720
```

16

语言定义举例

字符串 $0.5*X1+C$

词法 常数0.5; 算符*; 标识符X1; 算符+; 标识符C

语法 表达式 $0.5*X1+C$

语义 如果X1为非双精度型变量则先进行类型转换然后乘; 如果C为非双精度型变量则先进行类型转换然后加; 结果转换成单精度型。

17

语言定义举例

字符串 `fact(int n){if(n<=0){return 1}else{return n*fact(n-1)}};`

词法 标识符fact; 括号(; 保留字int; 标识符n; 括号); 括号{; 保留字if, ...

语法 函数定义`fact(int n){...}`

语义 函数定义缺省类型为整型; 参数结合规则; 函数副作用规则; 返回值类型转换规则...

如何精确定义一个语言?

18

2.2 高级语言的一般特性

数据类型 用于描述数据

整型、实型、双精度型、字符型、布尔型、数组、结构、类...

初等数据类型; 复合数据类型; 抽象数据类型

语句 用于描述功能/加工

赋值语句、循环语句、条件语句、说明语句、调用语句、返回语句...

语言分类 语言的范型

强制式Imperative、应用式Applicative、基于规则Rule-Based、面向对象Object-Oriented

19

C type	Size (bytes)	Lower bound	Upper bound
char	1	—	—
unsigned char	1	0	255
short int	2	-32768	+32767
unsigned short int	2	0	65536
(long) int	4	-2^{31}	$+2^{31} - 1$
float	4	-3.2×10^{-98}	$+3.2 \times 10^{98}$
double	8	-1.7×10^{-308}	$+1.7 \times 10^{308}$

C type	Pascal equivalent
char	char
unsigned char	—
short int	integer
unsigned short int	—
long int	longint
float	real
double	extended

初等数据类型用于描述数据
`char ch;`
`int book_no;`
`float amount;`

20

复合数据类型用于描述数据结构

抽象数据类型用于描述数据和代码的封装、继承、多态

```
record
    year : 0..2000;
    month : 1..12;
    day : 1..31
end

record
    name, firstname : string;
    age : 0..99;
    case married : Boolean of
        true : (Spousesname : string);
        false : ()
end
```

21

程序结构: Fortran语言

```
program main
some declarations
real alpha, beta
common /coeff/ alpha, beta
statements
stop
end
subroutine sub1 (some arguments)
declarations of arguments
real alpha, beta
common /coeff/ alpha, beta
statements
return
end
subroutine sub2 (some arguments)
declarations of arguments
real alpha, beta
common /coeff/ alpha, beta
statements
return
end
```

PROGRAM MAIN

...
END

SUBROUTINE SUB1

...
END

SUBROUTINE SUB2

...
END

22

程序结构: Pascal语言

```
program ScopeDemo;
var A : integer;
procedure Scopelnner;
var A : integer;
begin
    A := 10;
    writeln(A)
end;

begin (* Main *)
    A := 20;
    writeln(A);
    Scopelnner;
    writeln(A);
end. (* Main *)
```

23

Program Scope:

A, B, C

```
procedure Alpha;
A, F, G
```

```
procedure Beta;
VCR, Betamax, cassette
```

```
procedure Beta1;
Failure
```

```
function Beta2;
FailureToo
```

24

程序结构: C语言

- 预处理命令
- 类型说明
- 函数原型声明
- 变量说明
- 函数定义

```
type fn (parameters)
{
    local-vars;
    statements;
}
```

```
void c_function1(int a, float *b)
{
    .....
}

int c_function2(int a, float *b)
{
    int r;
    .....
    return r;
}

int main(void)
{
    int a, r;
    float b;
    c_function1(a, &b);
    r = c_function2(a, &b);
}
```

语言分类

- 强制式语言Fortran,C,Pascal,Ada等
- 应用式语言LISP,ML等
- 基于规则语言Prolog等
- 面向对象语言Smalltalk,C++,Java等

语句序列
函数调用
规则序列
类和实例

```
nrev([],[]).
nrev([A|X],Y):-nrev(X,Z),append(Z,[A],Y).
append([],X,X).
append([T|C],Y,[T|Z]):-append(C,Y,Z).

:-nrev([1,2,3,4,5],L)).
L = [5,4,3,2,1]
Continue search?(Y or N): Y
no
```

26

2.3 程序设计语言的语法描述

- 文法引论
- 上下文无关文法
- 语法分析树
- 文法的二义性问题
- 形式语言简介

27

2.3.1 文法引论

 $\Sigma = \{a, b, \#\}$

- 基本术语 $\Sigma^* = \{\epsilon, a, b, \#, aa, ab, a\#, ba, bb, b\#, aaa, \dots\}$

有穷字母表 Σ : 符号的有穷集合

Σ 上的符号串: Σ 中的一个有穷序列

空符号串/空字 ϵ : 不包含任何符号的字符串

Σ^* : Σ 上所有字符串的全体

字符串集合 U : $U \subseteq \Sigma^*$

28

■ 基本术语

符号串集合的连接(积):

$$UV = \{\alpha\beta \mid \alpha \in U \wedge \beta \in V\}, U \subseteq \Sigma^*, V \subseteq \Sigma^*$$

结合律: $(UV)W = U(VW)$

无交换律: $UV \neq VU$

$V^n = VV\dots V$ 自身的n次连接的连接(积)

$$V^0 = \{\epsilon\}$$

V 的闭包: $V^* = V^0 \cup V^1 \cup V^2 \cup \dots$

V 的正则闭包: $V^+ = VV^*$

29

举例

$$\Sigma = \{a, b, \#\}$$

$$\Sigma^* = \{\epsilon, a, b, \#, aa, ab, a\#, ba, bb, b\#, aaa, \dots\}$$

$$U = \{\epsilon, a, \#, aa\}$$

$$V = \{ab, a\#, bb\}$$

$$UV = \{ab, a\#, bb, aab, aa\#, abb, \#ab, \#a\#, \#bb, aaab, aaa\#, aabb\}$$

$$VU = \{ab, aba, ab\#, abaa, a\#, a\#a, a\#\#, a\#aa, bb, bba, bb\#, bbaa\}$$

$$U^2 = \{\epsilon, a, \#, aa, a\#, aaa, \#a, \#\#, \#aa, aa\#, aaaa\}$$

30

文法的规则

- 准确
- 易于理解
- 描述能力强



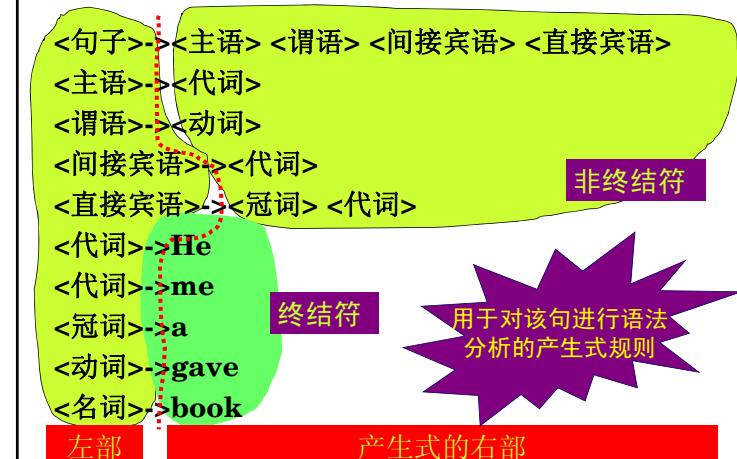
词法分析



语法分析

31

自然语言中的例子: He gave me a book.



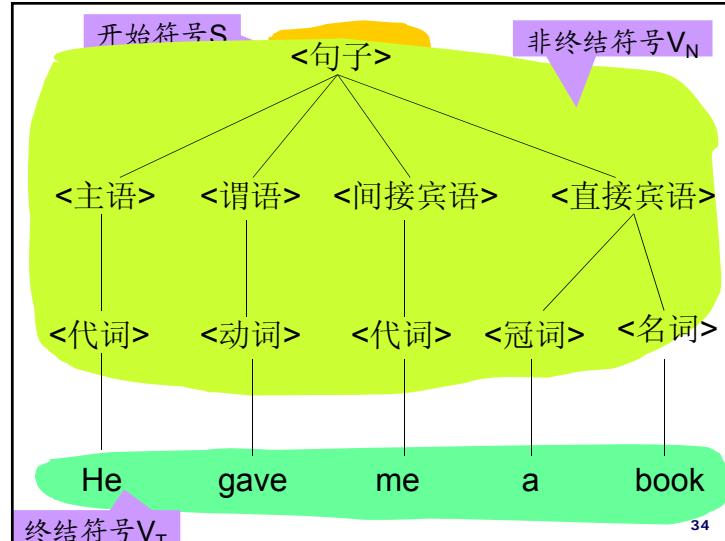
推导: He gave me a book.

```

<句子>=><主语> <谓语> <间接宾语> <直接宾语>
=><代词> <谓语> <间接宾语> <直接宾语>
=>He <谓语> <间接宾语> <直接宾语>
=>He <动词> <间接宾语> <直接宾语>
=>He gave <间接宾语> <直接宾语>
=>He gave <代词> <直接宾语>
=>He gave me <直接宾语>
=>He gave me <冠词> <名词>
=>He gave me a <名词>
=>He gave me a book

```

33



2.3.2上下文无关文法

一个文法 \mathbf{G} 是一个四元组: $\mathbf{G} = (V_T, V_N, S, \mathcal{P})$

- 终结符集合 V_T 是终结符号的非空有限集
 - ⊕ 终结符号是组成语言的基本符号, 不可再分解
- 非终结符集合 V_N 是非终结符号的非空有限集
 - ⊕ 非终结符号用来代表语法范畴
- 开始符号 S
 - ⊕ 开始符号是一个特殊的非终结符号, 代表语言中最感兴趣的语法范畴
- 产生式集合 \mathcal{P}
 - ⊕ 产生式规则是定义语法范畴的一种书写规则

35

上下文无关文法

文法 $\mathbf{G} = (V_T, V_N, S, \mathcal{P})$

$$\oplus V_T \cap V_N = \varphi$$

$$\oplus \text{若 } (P, \alpha) \in \mathcal{P}, \text{ 则 } P \in V_N \wedge \alpha \in (V_T \cup V_N)^*$$

⊕ 开始符号 S 至少必须在某个产生式的左部出现一次

36

产生式规则的书写形式

- ①大写拉丁字母表示非终结符
- ②小写拉丁字母表示终结符
- ③小写希腊字母表示由终结符和非终结符构成的符号串

$A \rightarrow \alpha$

左部为一个
非终结符

右部是符号串,
由终结符和非
终结符组成

$$\begin{aligned} E &\rightarrow i \\ E &\rightarrow E+E \\ E &\rightarrow E^*E \\ E &\rightarrow (E) \end{aligned}$$

37

产生式规则

$$\begin{aligned} P &\rightarrow \alpha_1 \\ P &\rightarrow \alpha_2 \\ &\vdots \\ P &\rightarrow \alpha_n \end{aligned} \quad \Rightarrow \quad P \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$$

$$\begin{aligned} E &\rightarrow i \\ E &\rightarrow E+E \\ E &\rightarrow E^*E \\ E &\rightarrow (E) \end{aligned} \quad \Rightarrow \quad E \rightarrow i | E+E | E^*E | (E)$$

38

推导与直接推导

对于 $(A, \gamma) \in \mathcal{P}$ 且 $\alpha, \beta \in (V_N \cup V_T)^*$,
符号串 $\alpha A \beta$ 使用一次规则 $A \rightarrow \gamma$ 得到
 $\alpha \gamma \beta$, 称 $\alpha A \beta$ 直接推导出 $\alpha \gamma \beta$, 记为
 $\alpha A \beta \Rightarrow \alpha \gamma \beta$

例: $E \rightarrow i | E+E | E^*E | (E)$

$E \Rightarrow (E)$ 是直接推导, 其中 $\alpha = \beta = \epsilon$

$E \Rightarrow i+E$ 不是直接推导

$E+E \Rightarrow i+E$ 是直接推导, 其中 $\alpha = \epsilon$, $\beta = +E$

39

推导与直接推导

对于 $\alpha_1, \alpha_2, \dots, \alpha_n \in (V_N \cup V_T)^*$,
如果 $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$
则称其为从 α_1 到 α_n 的一个推导.

例: $E \rightarrow i | E+E | E^*E | (E)$

$E \Rightarrow (E)$ 是推导

$E \Rightarrow E+E \Rightarrow i+E$ 是推导

$E+E \Rightarrow i+E$ 是推导

40

推导与直接推导

例: $E \rightarrow i \mid E+E \mid E^*E \mid (E)$

$E \Rightarrow E+E \Rightarrow E+E^*E \Rightarrow E+E^*(E) \Rightarrow E+E^*(E+E) \Rightarrow E+E^*(E+i) \Rightarrow E+E^*(i+i) \Rightarrow E+i^*(i+i) \Rightarrow i+i^*(i+i)$

$E \Rightarrow E+E \Rightarrow i+E \Rightarrow i+E^*E \Rightarrow i+i^*E \Rightarrow i+i^*(E) \Rightarrow i+i^*(E+E) \Rightarrow i+i^*(i+E) \Rightarrow i+i^*(i+i)$

对于推导 $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \dots \Rightarrow \alpha_n$, 若其中的直接推导步数不少于1, 则记为 $\alpha_1 \xrightarrow{+} \alpha_n$, 否则记为 $\alpha_1 \xrightarrow{*} \alpha_n$

41

句子、句型和语言

文法 G_1
 $S \rightarrow bA$
 $A \rightarrow aA \mid a$

$L(G_1) = \{ba^n \mid n \geq 1\}$

文法 G_2
 $S \rightarrow AB$
 $A \rightarrow aA \mid a$
 $B \rightarrow bB \mid b$

$L(G_2) = \{a^m b^n \mid m, n \geq 1\}$

假定文法G的开始符号为S,
如果 $S \xrightarrow{*} \alpha$ 则称 α 是一个句型
仅含终结符号的句型称为句子
句子的全体是一个语言, 记为 $L(G)$

$L(G) = \{\alpha \mid S \xrightarrow{+} \alpha \wedge \alpha \in V_T^*\}$

42

最左推导与最右推导

最左(右)推导是一种推导, 其中每一步直接推导中都是对最左(右)边的非终结符号使用产生式规则进行替换得到。

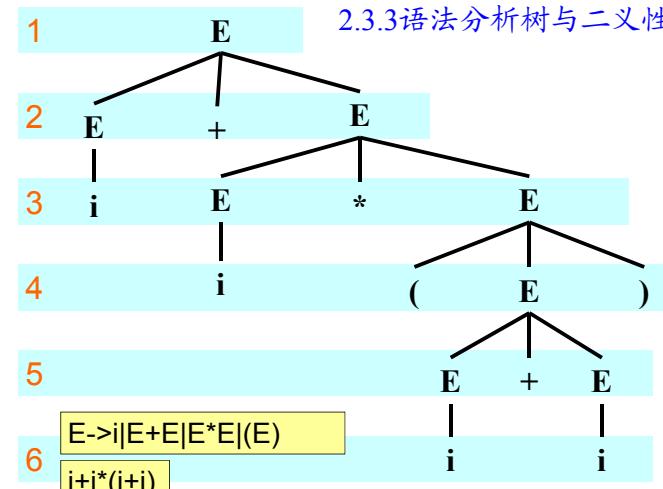
例: $E \rightarrow i \mid E+E \mid E^*E \mid (E)$

最左推导 $E \Rightarrow E+E \Rightarrow i+E \Rightarrow i+E^*E \Rightarrow i+i^*E \Rightarrow i+i^*(E) \Rightarrow i+i^*(E+E) \Rightarrow i+i^*(i+E) \Rightarrow i+i^*(i+i)$

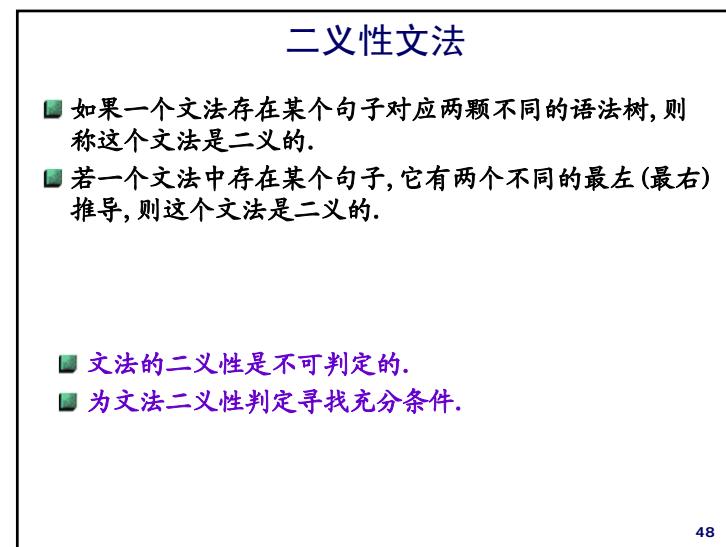
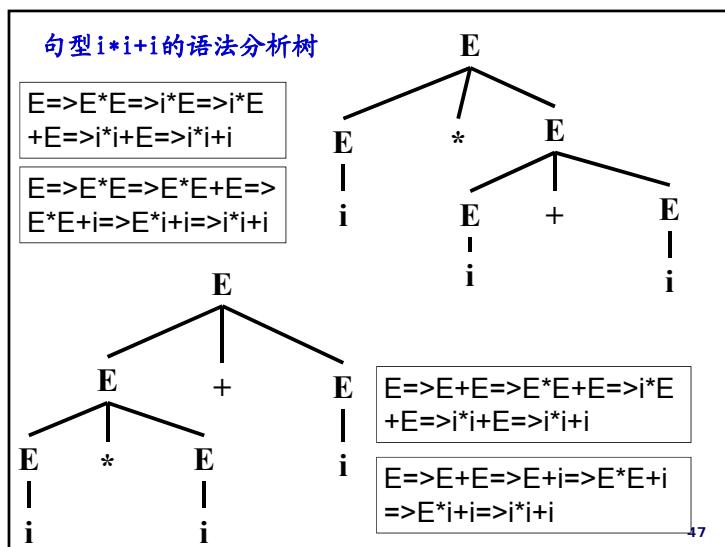
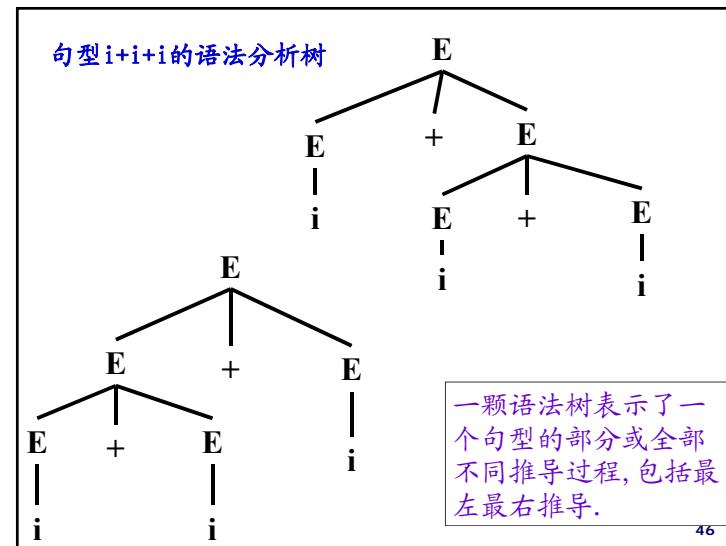
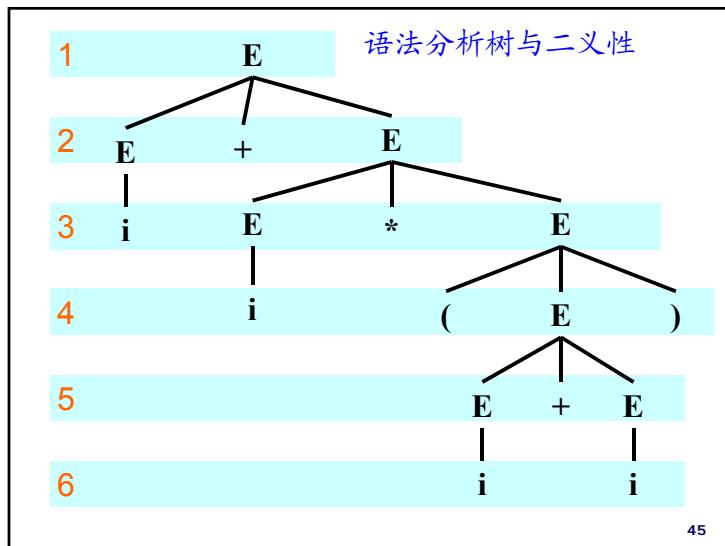
最右推导 $E \Rightarrow E+E \Rightarrow E+E^*E \Rightarrow E+E^*(E) \Rightarrow E+E^*(E+E) \Rightarrow E+E^*(E+i) \Rightarrow E+E^*(i+i) \Rightarrow E+i^*(i+i) \Rightarrow i+i^*(i+i)$

43

2.3.3 语法分析树与二义性



44



文法G2是在文法G1的基础上
加了如下归约限定:

- ① 连+从左到右
- ② 连*从左到右
- ③ 先*后+

- ①对于 $i_1+i_2+i_3$ 先归约 i_1+i_2
- ②对于 $i_1^*i_2^*i_3$ 先归约 $i_1^*i_2$
- ③对于 $i_1^*i_2+i_3$ 先归约 $i_1^*i_2$
- ④对于 $i_1+i_2^*i_3$ 先归约 $i_2^*i_3$

文法G1:

$$E \rightarrow i \mid E+E \mid E^*E \mid (E)$$

文法G2:

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

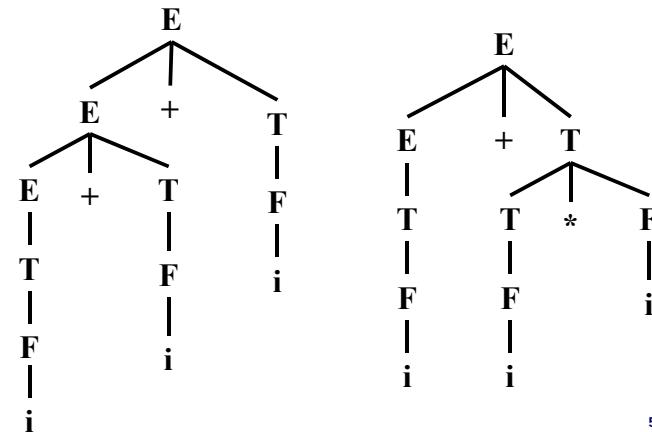
$$T \rightarrow F \mid T^*F$$

$$F \rightarrow (E) \mid i$$

结论: $L(G1)=L(G2)$

49

无二义性文法的任意句子的语法树是唯一的,
所以该句子的最左推导唯一。



50

约定

- 以后均讨论无二义性文法，同时有下面的约定:
 - ⊕ 文法中不含形如 $P \rightarrow P$ 的产生式规则；
 - ⊕ 每个非终结符必须都有用处，即对于任一非终结符 P ，有

$$S \xrightarrow{*} \alpha P \beta \wedge (\exists \gamma \in V_T^*) P \xrightarrow{+} \gamma$$

51

2.3.3 形式语言简介

Chomsky文法

$$G = (V_t, V_N, S, \mathcal{P})$$

0型文法

 $\forall (\alpha, \beta) \in \mathcal{P}$ 有:

$$\alpha \rightarrow \beta, \alpha \in (V_N \cup V_T)^*, \beta \in (V_N \cup V_T)^*$$

1型文法

G的任何产生式 $\alpha \rightarrow \beta$ 均满足 $|\alpha| \leq |\beta|$, 仅 $S \rightarrow \epsilon$ 除外, 且 β 中不含 S ;2型文法 $A \rightarrow \beta, A \in V_N, \beta \in (V_N \cup V_T)^*$ 3型文法 $A \rightarrow \alpha B, \text{或}, A \rightarrow \alpha, \text{且} A, B \in V_N, \alpha \in (V_T)^*$

52

本章小结

- 字母表、符号串
- 终结符号, 非终结符号, 开始符号, 产生式
- 直接推导、推导
- 句型、句子
- 语言
- 语法树、规约
- 二义性文法
- Chomsky文法

53

本章练习

- 了解高级语言的主要内容及特点;
- 掌握上下文无关文法及语法树。
- 作业: p35-36: 4, 6-10.

54