第二章 高级语言及其语法描述

- 2.1 程序设计语言的定义
- 2.2 高级语言的一般特性
- 2.3 程序设计语言的语法描述

本章目的:

•简要了解高级语言的主要内容及特点;

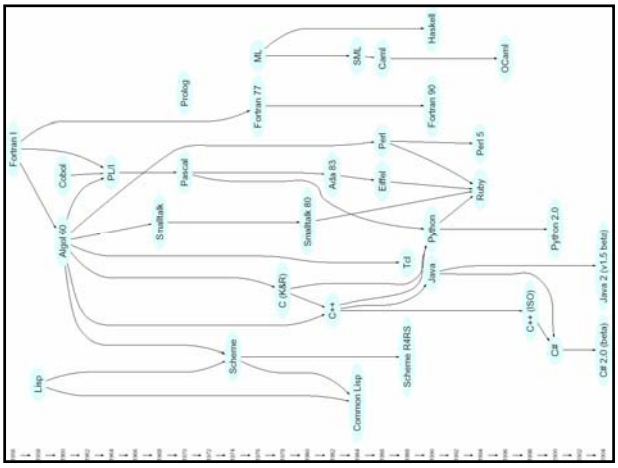
•掌握上下文无关文法及语法树。

作业: p35-36:1(1)(2)(5),4,6-11。9号交

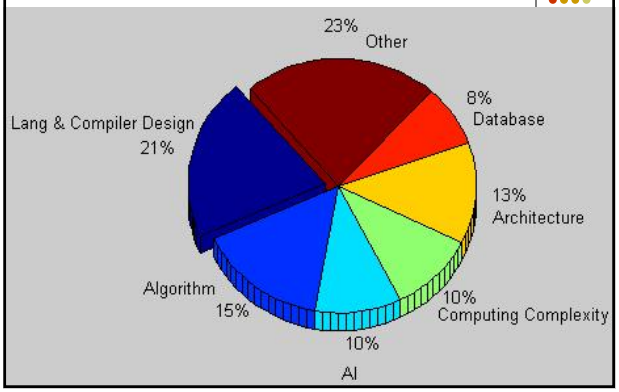
程序设计语言历史

- 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.



A.M. Turing Award



2.1 程序设计语言的定义

- 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

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

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。

语言定义举例

符号串 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？

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
  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

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	$-3.2\times10^{\pm38}$	$+3.2\times10^{\pm38}$
double	8	$-1.7\times10^{\pm308}$	$+1.7\times10^{\pm308}$

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;
```

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
```

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
```

程序结构: Pascal语言

```
program ScopeDemo;
var A : integer;

 procedure ScopeInner;
 var A : integer;
 begin
   A := 10;
   writeln (A)
 end;

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

```
program main
…
        procedure p1;
        …
                procedure p2;
                …
                begin
                …
                end;
        begin
        …
        end;
        …
begin
…
end.
```

Program Scope:
A, B, C

procedure Alpha;
A, F, G

procedure Beta;
VCR, Betamax, cassette

procedure Beta1;
Failure

function Beta2;
FailureToo

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
  语句序列
- 应用式语言
  函数调用
- 基于规则语
  规则序列
- 面向对象语
  类和实例

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

>(fact 6)
720
```

```
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
```

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

Σ={a, b, #}

Σ*={ ε , a, b, #, aa, ab, a#, ba, bb, b#, aaa, …}

基本术语

有穷字母表Σ：	符号的有穷集合
Σ上的符号串：	Σ中的符号的一个有穷序列
空符号串/空字 ε ：	不包含任何符号的符号串
Σ*：	Σ上所有符号串的全体
符号串集合U：	$U \subseteq \Sigma^*$

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：$UV \neq VU$

$V^n = VV...V$ 自身的n次连接的连接(积)

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

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

V的正则闭包：$V^+ = VV^*$

举例

Σ={a, b, #}

Σ^*={ ε , a, b, #, aa, ab, a#, ba, bb, b#, aaa, …}

U={ ε , 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={ ε , a, #, aa, a#, aaa, #a, ##, #aa, aa#, aaaa}

自然语言语法分析的例子

语法分析：He gave me a book.

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

非终结符

终结符

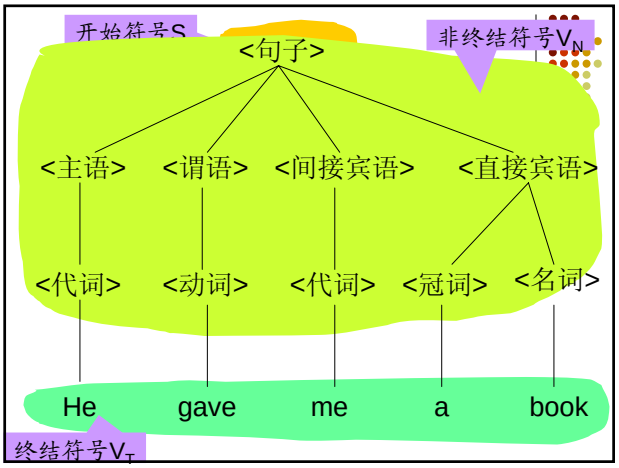
用于对该句进行语法分析的产生式规则

左部

产生式的右部

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



2.3.2上下文无关文法

一个文法G是一个四元组: $G=(V_T,V_N,S,P)$

❖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P

产生式规则是定义语法范畴的一种书写规则

上下文无关文法

文法 $G=(V_T,V_N,S,P)$

●$V_T \cap V_N = \Phi$

●若$(P, \alpha) \in P$，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

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

$$A \to \alpha$$

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

E->i
E->E+E
E->E*E
E->(E)

产生式规则

$$P \to \alpha_1$$
$$P \to \alpha_2$$
$$\vdots$$
$$P \to \alpha_n$$

$$\Rightarrow \quad P \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$$

每个 α_i 称为P的候选式，i=1..n

E->i
E->E+E
E->E*E
E->(E)

⇒ E->i|E+E|E*E|(E)

推导与直接推导

对于 $(A,\gamma) \in P$ 且 $\alpha,\beta \in (V_N \cup V_T)^*$，字符串 $\alpha A\beta$ 使用一次规则 $A \to \gamma$ 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->i|E+E|E*E|(E)

E=>(E)　是直接推导，其中 α = β = ε
E=>i+E　不是直接推导
E+E=>i+E　是直接推导，其中 α = ε，β =+E

推导与直接推导

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

例：E->i|E+E|E*E|(E)

E=>(E)　是推导
E=>E+E=>i+E　是推导
E+E=>i+E　是推导

推导与直接推导

例：E->i|E+E|E*E|(E)

E=>E+E=>E+E*E=>E+E*(E)=>E+E*(E+E)=>E+E*(E+i)=>E+E*(i+i)=>E+i*(i+i)=>i+i*(i+i)

E=>E+E=>i+E=>i+E*E=>i+i*E=>i+i*(E)=>i+i*(E+E)=>i+i*(i+E)=>i+i*(i+i)

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

句子、句型和语言

文法 G_1
S->bA
A->aA|a

$L(G_1)=\{ba^n|n>=1\}$

文法 G_2
S->AB
A->aA|a
B->bB|b

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

假定文法G的开始符号为S，

如果 $S \overset{*}{\Rightarrow} \alpha$ 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)

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

最左推导与最右推导

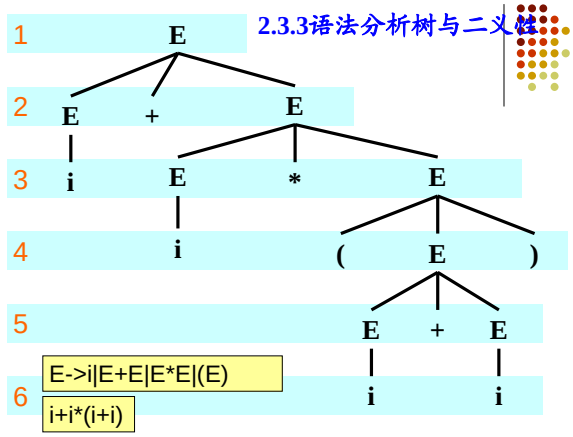
最左（右）推导是一种推导，其中每一步直接推导中都是对最左（右）边的非终结符号使用产生式规则进行替换得到。

例：E->i|E+E|E*E|(E)

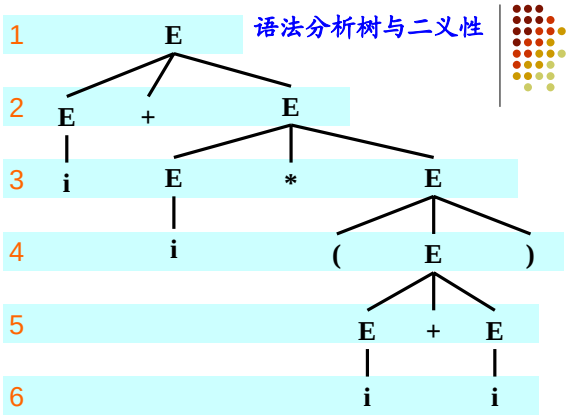
最左推导 E=>E+E=>i+E=>i+E*E=>i+i*E=>i+i*(E)=>i+i*(E+E)=>i+i*(i+E)=>i+i*(i+i)

最右推导 E=>E+E=>E+E*E=>E+E*(E)=>E+E*(E+E)=>E+E*(E+i)=>E+E*(i+i)=>E+i*(i+i)=>i+i*(i+i)

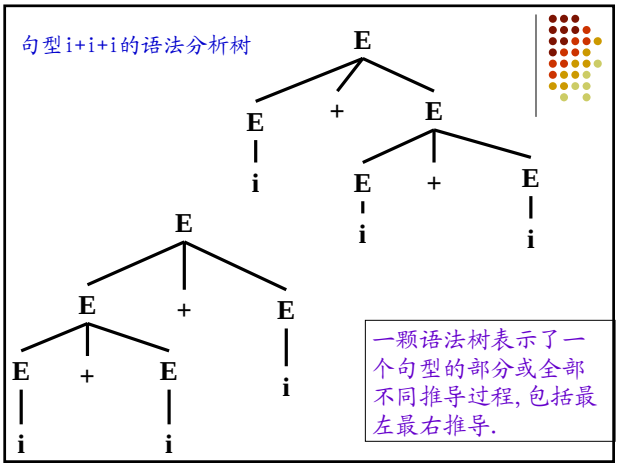
2.3.3语法分析树与二义性



语法分析树与二义性



句型i+i+i的语法分析树

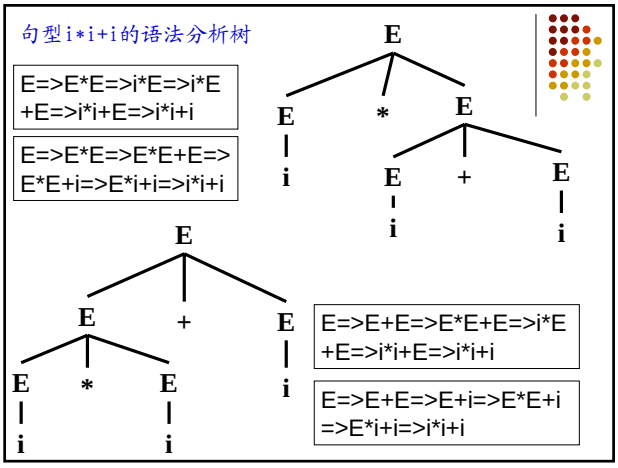


一颗语法树表示了一个句型的部分或全部不同推导过程，包括最左最右推导.

句型i*i+i的语法分析树

E=>E*E=>i*E=>i*E+E=>i*i+E=>i*i+i

E=>E*E=>E*E+E=>E*E+i=>E*i+i=>i*i+i



E=>E+E=>E*E+E=>i*E+E=>i*i+E=>i*i+i

E=>E+E=>E+i=>E*E+i=>E*i+i=>i*i+i

二义性文法

- 如果一个文法存在某个句子对应两颗不同的语法树，则称这个文法是二义的.
- 若一个文法中存在某个句子，它有两个不同的最左（最右）推导，则这个文法是二义的.

- 文法的二义性是不可判定的.
- 为文法二义性判定寻找充分条件.

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->i|E+E|E*E|(E)

文法G2:

E->T|E+T

T->F|T*F

F->(E)|i

结论: L(G1)=L(G2)

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。

总结:

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->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

$$S \overset{*}{\Rightarrow} \alpha P \beta \wedge (P \overset{+}{\Rightarrow} \gamma, \gamma \in V_T^*)$$

i

2.3.3 形式语言简介

Chomsky文法

$G = (V_t, V_N, S, P)$

0型文法 $\forall(\alpha,\beta) \in P$ 有:

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

1型文法 G的任何产生式 α → β 均满足| α |<=| β |, 仅S→ ε除外，且 β 中不含S;

2型文法 $A \to \beta, A \in V_N, \beta \in (V_N \cup V_T)^*$

3型文法 $A \to \alpha B$,或, $A \to \alpha$,且$A, B \in V_N, \alpha \in (V_T)^*$

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