第3章 形式语言与语法分析

- 3.1 文法
- 3.2 语法分析概述

3.1 文法

- 3.1.1 文法的直观概念
- 3.1.2 文法和语言的形式定义
- 3.1.3 文法的类型*
- 3.1.4 上下文无关文法及其语法树
- 3.1.5 有关文法实用中的一些说明*

3.1.1 文法的直观概念

- <句子>::=<主语><谓语>
- <主语>::=<代词>|<名词>
- <代词>::=我|你|他
- <名词>::=王明|大学生|工人|英语
- <谓语>::=<动词><直接宾语>
- <动词>::=是|学习
- <直接宾语>::=<代词>|<名词>
- “我是大学生”符合上述规则，是句子
- “我大学生是”不符合上述规则，不是句子
- 符号"⇒"的含义是使用一条规则，代替"⇒"左端的某个符号，产生"⇒"右端的符号串。
- 应用"⇒"可以生成"王明是大学生"、"我学习英语"、"你是英语"、“你学习工人”等句子。
- 文法是以有穷的集合刻画无穷集合的一个工具。

3.1.2 文法和语言的形式定义

- 文法： 文法G是一个四元组G=(N,T,P,S)，其中：
  - N: 非终极符号(Nonterminals)的有穷集合,不可为空；
  - T: 终极符号(Terminals)的有穷集合,不可为空,

    且 $N \cap T = \Phi$；（通常用V表示 $N \cup T$）
  - S: 文法开始符号，$S \in N$；
  - P: 产生式(Productions)的有穷集合,

  一般形式 $\alpha \to \beta$ 或 $\alpha ::= \beta$,读作:" α 产生出 β "或:" α 推出 β "； 其中 $\alpha, \beta \in V^*$ ，α 称为产生式的左部，β 称为产生式的右部，α 不能为空.

3.1.2 文法和语言的形式定义

- 例子3.1： 文法G=(N,T,P,S), 其中N={W}, T={0,1}, P={W→0W1, W →01}, S = W.
- 例子3.2： 文法G=(N,T,P,S), 其中N={标识符,字母,数字}, T={a,b,c…,x,y,z,0,1,…,9}, P=
- {<标识符>→<字母>
  <标识符>→<标识符><字母>
  <标识符>→<标识符><数字>
  <字母> →a
  <字母> →b
  ……
  <字母> →z
  <数字> →0
  <数字> →1
  ……
  <数字> →9 }
  S = <标识符>

※< >括起非终极符

※相同左部的多条产生式可以合并. 例1的产生式可写作:

W → 0W1 | 01

※G=(N,T,P,S),可以写作G[S],表明S是文法G的开始符

※通常省略四元组， 只用产生式来定义文法,则约定第一条产生式的左部是开始符

3.1.2 文法和语言的形式定义

- 引入"推导"的概念,定义V^*中符号之间的关系:直接推导$\Rightarrow$、长度为$n(n\geq1)$的推导$\Rightarrow^+$和长度为$n(n\geq0)$的推导$\Rightarrow^*$.
- 直接推导: 如$\alpha\rightarrow\beta$是文法$G=(N,T,P,S)$的规则(或说是P中的一条产生式),γ和δ是V^*中的任意符号,若有符号串ν,ω满足$\nu=\gamma\alpha\delta$, $\omega=\gamma\beta\delta$, 则说ν(应用规则$\alpha\rightarrow\beta$)直接产生ω,或说, ω是ν的直接推导,或说, ν是ω的直接归约, 记作$\nu\Rightarrow\omega$.
- 推导$\Rightarrow^+$: 若存在直接推导的序列:

  $$\nu=\omega_0\Rightarrow\omega_1\Rightarrow\omega_2\ldots\omega_n=\omega\ (n>0)$$

  则称ν推导出ω, 或称ω归约出ν, 记作$\nu\Rightarrow^+\omega$。
- 推导$\Rightarrow^*$: 若有$\nu\Rightarrow^+\omega$, 或$\nu=\omega$, 则记作$\nu\Rightarrow^*\omega$.

3.1.2 文法和语言的形式定义

- 对于例子3.1的文法G，可以给出直接推导的一些例子如下：
- (1) 0W1 ⇒ 0011, 使用规则W →01
- (2) 0W1 ⇒ 00W11, 使用规则W→0W1
- 对于例子3.2的文法G，直接推导的一些例子如下：
- (1) <标识符> ⇒ <标识符><字母>,使用规则<标识符>→<标识符><字母>
- (2) <标识符><字母><数字> ⇒ <字母><字母><数字> ,使用规则<标识符>→<字母>
- (3) <字母><字母><数字> ⇒a <字母><数字> ,使用规则<字母> →a
- 思考: <字母><字母><数字> ⇒a b<数字>?

3.1.2 文法和语言的形式定义

- 对于例子3.1的文法G，可以给出推导的例子如下：

  $0W1 \Rightarrow 00W11 \Rightarrow 000W111 \Rightarrow 0000111$,

  即$0W1 \Rightarrow^{+} 0000111$，也可记作

  $0W1 \Rightarrow^{*} 0000111$

- 对于例子3.2的文法G，推导的例子如下：

  (1) <标识符> ⇒ <标识符><字母> ⇒<标识符><字母><数字> ⇒ <字母><字母><数字> ⇒a <字母><数字> ⇒ab<数字> ⇒abc

  即<标识符> $\Rightarrow^{+}$abc, 也可记作<标识符> $\Rightarrow^{*}$abc

  (2) <标识符> $\Rightarrow^{*}$ <标识符>　　√

  (3) <标识符> $\Rightarrow^{+}$ <标识符>　　×

3.1.2 文法和语言的形式定义

- 句型：如果有$S \stackrel{*}{\Rightarrow} \alpha$, S是文法的开始符, $\alpha \in V^*$，则称α是G的句型
  - W,0W1,000111,00001111都是例3.1中文法的句型
  - <标识符>,<字母>,<标识符><字母>，a1,a1a都是例3.2中文法的句型
  - 0101,1w0, 1100是例3.1中文法的句型吗？ NO!
  - 1<字母>, <数字><字母>是例3.2中文法的句型吗? NO!
- 句子：如果有$S \stackrel{+}{\Rightarrow} \alpha$, S是文法的开始符, $\alpha \in T^*$，则称α是G的句子
  - 000111,00001111是例3.1中文法的句子
  - a1,a1a是例3.2中文法的句子

3.1.2 文法和语言的形式定义

- 语言： $L(G) = \{\alpha \mid S \Rightarrow^{+} \alpha, \alpha \in T^{*}\}$,即文法G的所有句子的集合
  - 例3.1中文法定义的语言L(G) = $\{0^n1^n \mid n \geq 1\}$
  - 例3.2中文法定义的句子是字母开头、字母和数字字符构成的串。文法定义的语言是程序设计语言中用于表示名字的标识符。
- 文法等价： G_1和G_2两个文法,若$L(G_1) = L(G_2)$,则称G_1和G_2等价
  - $A \rightarrow 0R$
  - $A \rightarrow 01$
  - $R \rightarrow A1$　　与例3.1的文法等价

3.1.3 文法的类型*

- 乔姆斯基(Chomsky)把文法分成四种类型,即0型，1型，2型和3型。差别在于对产生式施加不同的限制。
- 设G=(N,T,P,S)，如果它的每个产生式$\alpha \to \beta$是这样一种结构:
- (1) $\alpha \in V^*$,且至少含有一个N, $\beta \in V^*$,则G是0型文法,也称短语文法;
- (2) $|\beta| \geq |\alpha|$,仅仅$S \to \varepsilon$除外,则G是1型文法,也称上下文有关文法(context-sensitive);
- (3) $\alpha \in N$, $\beta \in V^*$,则G是2型文法,也称上下文无关文法(context-free);
- (4) 若P中的每个产生式的形式都是$A \to aB$或$A \to a$,其中$A,B \in N$, $a \in T^*$,则G是3型文法,也称正则文法

3.1.4 上下文无关文法及其语法树

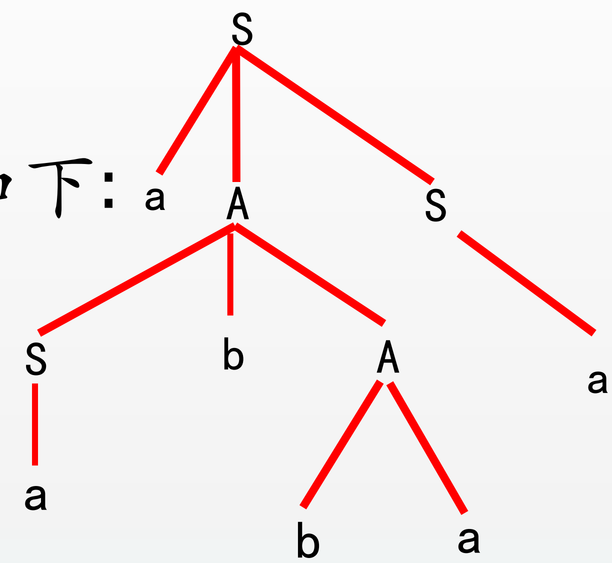
- 上下文无关文法有足够的能力描述现今程序设计语言的语法结构.
- 例3.3 文法G = ({E},{+,*,i,(,)},P,E),其中P为:
- E→i E →E+E   E →E*E E →(E)
- 例3.4 文法片段:
- <赋值语句> →i = <E>
- <条件语句> →if <条件> <语句>
-                         | if <条件> <语句> else <语句>
- 如无特别说明，下文中出现的”文法”一词均指上下文无关文法

3.1.4 上下文无关文法及其语法树

- 语法树(推导树):一种表示句型推导的直观工具
- 定义：给定文法G= (N,T,P,S),对于G的任何句型都能构造与之关联的语法树. 这棵树满足下列4个条件:
- (1) 每个结点都有一个标记,此标记是V中的一个符号;
- (2) 根的标记是S;
- (3) 若一结点node(标记为A)至少有一个它自己除外的子孙, 则$A \in N$;
- (4) 如果结点node (标记为A)的直接子孙,从左到右的次序是结点$n_1, n_2, \ldots, n_k$,其标记分别为$A_1, A_2, \ldots, A_k$,那么$A \to A_1 A_2 \ldots A_k$一定是P中的产生式.

3.1.4 上下文无关文法及其语法树

- 语法树(推导树)的例子:
- 例3.5 文法G的产生式P如下:
  - $S \rightarrow aAS$
  - $A \rightarrow SbA$
  - $A \rightarrow SS$
  - $S \rightarrow a$
  - $A \rightarrow ba$



- 语法树表示了在推导过程中施用了哪个产生式和施用在哪个非终极符上，它没有标明施用产生式的顺序。即一棵语法树表示了一个句型的种种可能的(未必是所有的)不同推导过程.

3.1.4 上下文无关文法及其语法树

- 最左(右)推导:如果在推导的任何一步$\alpha \Rightarrow \beta$,其中α, β是句型,都是对α中的最左(右)非终极符施用产生式进行替换,则称这种推导为最左(最右)推导.

- 例3.5 中文法G的句型aabbaa的推导过程可以有很多:
- 推导过程1: S $\Rightarrow$aAS $\Rightarrow$aSbAS $\Rightarrow$aabAS $\Rightarrow$aabbaS $\Rightarrow$aabbaa
  最左推导
- 推导过程2: S $\Rightarrow$aAS $\Rightarrow$aAa $\Rightarrow$aSbAa $\Rightarrow$aSbbaa $\Rightarrow$aabbaa.

  最右推导(规范推导)

- 规范句型: 由规范推导所得的句型称为规范句型.
- S,aAS,aAa,aSbAa,aSbbaa,aabbaa都是规范句型
- 思考: 一个句型是否只对应唯一的一棵语法树呢? 一个句型是否只唯一地对应一个最左(最右)推导呢? No!

3.1.4 上下文无关文法及其语法树

- 例3.6: 对于例3.3中的文法G，句型i+i*i就有两个不同的最左推导1和2，它们所对应的语法树分别如图1、图2所示：
- 推导1：$E \Rightarrow E+E \Rightarrow i+E \Rightarrow i+E*E \Rightarrow i+i*E \Rightarrow i+i*i$
- 推导2：$E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow i+E*E \Rightarrow i+i*E \Rightarrow i+i*i$
-

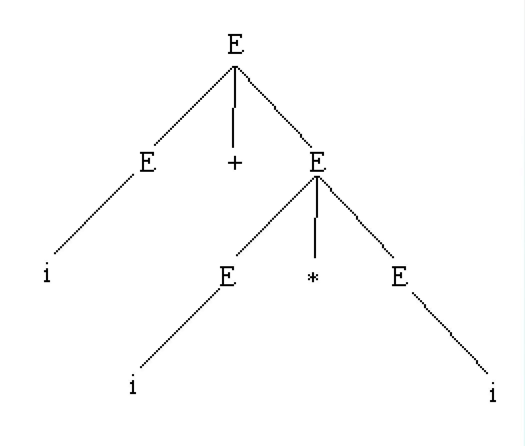


图1

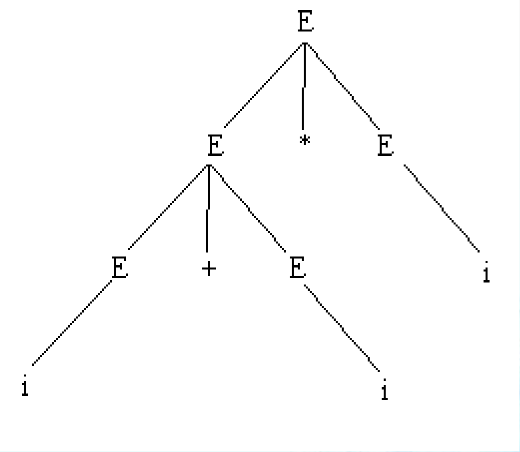


图2

3.1.4 上下文无关文法及其语法树

- 二义性文法: 如果一个文法存在某个句子对应两棵不同的语法树，或者如果一个文法存在某个句子存在两个不同的最左(最右)推导,则说这个文法是二义的.
- 二义性文法给语言的语句的分析带来一定的困难，因此常常希望文法是无二义的.
- 文法是非二义的是不可判定的.
- 文法是二义的可通过寻找二义性的句子证明.
- 对于非二义性语言，一定存在一个非二义性的文法定义.
- 例3.3中表达式的非二义性文法如下:
- $E \to T \mid E + T$
- $T \to F \mid T * F$
- $F \to (E) \mid i$

3.1.5 有关文法实用中的一些说明*

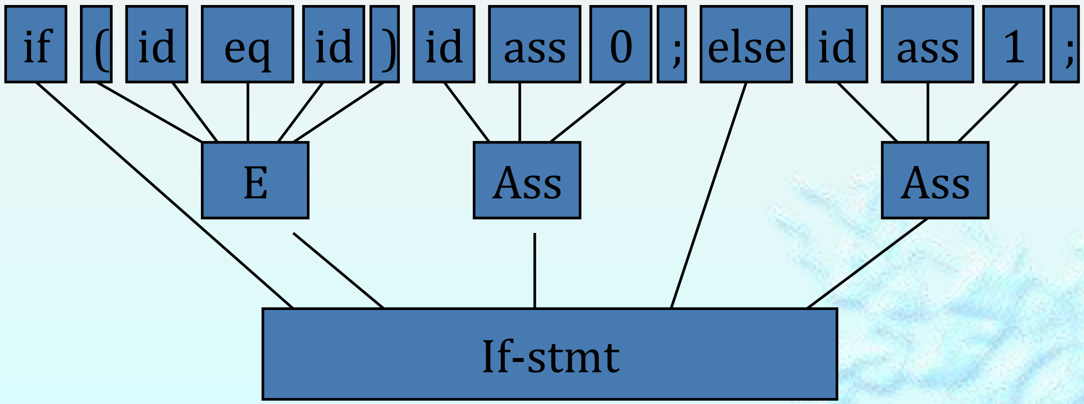
- 限制文法不得含有特型产生式*
  - 特型产生式: 形如U→ U
- 限制文法不得含有无用产生式*
  - 无用产生式: 没有一个句子的推导中用得到的
  - 从开始符无法到达的非终极符
  - 若从该非终极符无法推导出终极符串(意味着文法出错了！)
- 有时限制文法不得含有空产生式*
  - 空产生式: A → ε

3.2 语法分析概述

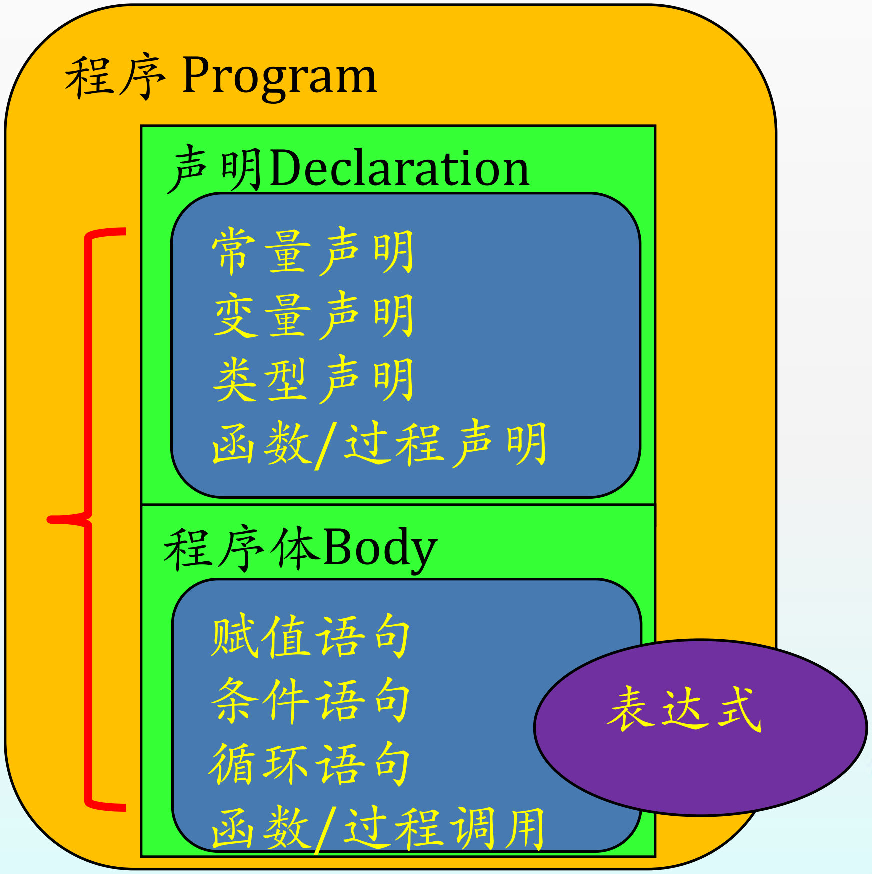
- 3.2.1 语法分析的任务
- 3.2.2 语法结构
- 3.2.3 语法错误
- 3.2.4 语法分析方法

3.2.1 语法分析的任务

- 语法分析：检查源程序的语法是否正确，并将其由Token序列形式转换成表示源程序语法结构的“语法树”形式，若有错误还会报告语法错误.
- 语法分析前:
- 语法分析后:



3.2.2 语法结构



程序 Program
声明Declaration
常量声明
变量声明
类型声明
函数/过程声明
程序体Body
赋值语句
条件语句
循环语句
函数/过程调用
表达式

3.2.3 语法错误

- 语法结构的开始单词错误
  - 表达式E的开始单词是id, (, n---其他Token则出错
- 语法结构的后继单词错误
  - 语句；语句----语句，语句
- 标识符或者常量错
  - id := E ---- begin := E 或 123:=E
- 关键字错
  - if E then 语句 else 语句 ----其他关键字则出错
- 括号配对错
  - （ () () ） ----) （ ()

3.2.4 语法分析方法

- 为方便起见，均为从左到右的分析
- 自顶向下分析: 从文法的开始符号出发,反复使用各种产生式,寻找"匹配"于输入符号串的推导
- 自底向上分析: 从输入符号串开始,逐步进行"归约",直至归约到文法的开始符号
- 例3.7: 文法G的产生式如下:
  - $S \rightarrow aABd$
  - $A \rightarrow b \mid bA$
  - $B \rightarrow c \mid cB$
  - $B \rightarrow d \mid b$
  - 分析符号串abcd是否是该文法的句子.