第二章
文法和语言

本章内容

2.1 符号、符号串及其集合的运算
2.2 文法的直观概念
2.3 文法和语言的形式定义
2.4 文法的类型
2.5 上下文无关文法及其语法树
2.6 句型的分析
2.7文法实用中的一些说明

2.1 符号、符号串及其集合的运算

任何一种语言都是由该语言的基本符号组组成的符号串的集合。

- 基本符号集
- 任何语言的单词符号就是定义在它的字符集上的字符串
- 该语言的任何语句就是定义在其单词符号集上的单词串(符号串)

2.1.1 字母表和符号串

2.1.2 符号串及其集合的运算

2.2.1 字母表和符号串

- 字母表：是元素的非空有穷集合，把字母表中的元素称为符号，因此字母表也称符号集。例
- 符号串：字母表中的符号串组成的任何有穷序列。例

2.1.2 符号串及其集合的运算

① 符号串的长度：如果某符号串x中有m个符号，则其长度为m，记为|x|=m。例

② 符号串的联接：设x和y是符号串，它们的联接xy是把y的符号写在x的符号之后得到的符号串。例

③ 符号串的方幂：设x是符号串，把x自身连接n次得到符号串z，即z=xxx….xx，称为符号串x的方幂。例

④ 符号串集合：若集合A中的一切元素都是某字母表上的符号串，则称A为字母表上的符号串集合。

- 语言：表示某个确定的字母表上的符号串的任何集合。

⑤集合的乘积：AB={xy|x∈A且y∈B},例。

⑥ 集合A的闭包A*和正闭包A⁺：

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \ldots \cup \Sigma^n \ldots$$
$$\Sigma^+ = \Sigma^1 \cup \ldots \cup \Sigma^n \ldots$$

$$\begin{cases} \Sigma^* = \Sigma^0 \cup \Sigma^+ \\ \Sigma^+ = \Sigma\Sigma^* = \Sigma^*\Sigma \end{cases}$$

2.2 文法的直观概念

文法的定义：对语言结构的描述和定义，即在形式上用来描述和规定语言结构的称为“文法”(或“语法”)。

例

规则

句子的语法结构，可以用一组规则来描述。

规则也称为“产生式”，规则中的“::=”也常用“→”表示。

注意：文法中，描述某个特定的语法成分的规则可能不只一条。

由规则推导句子

如果用一组规则来描述句子的结构，就可以利用这组规则按照一定的方式去推导产生句子。

推导方法：使用一条规则，代替=>左边的某个符号，产生=>右端的符号串

2.3 文法和语言的形式定义

前面已经对规则(或产生式)的概念进行了非形式化的说明，我们已经对其有了一个直观的了解。下面将对其进行形式化说明，并在此基础上抽象地定义文法和语言。

2.3.1 文法的形式定义

2.3.2 推导的形式定义

2.3.3 语言的形式定义

2.3.1 文法的形式定义

① 规则、产生式(或重写规则)：形如 $\alpha \rightarrow \beta$ 或 $\alpha ::= \beta$ 的（α，β）有序对，且 $\alpha \in V^+$，$\beta \in V^*$，V为某字母表。

- α 称为规则的左部(或产生式的左部)。
- β 称为规则的右部(或产生式的右部)。

②文法的定义

- 文法G定义为四元组（V_N，V_T，P，S）
  - V_N：非终结符集
  - V_T：终结符集
  - P：产生式（规则）集合
  - S：开始符号

$V_N \cap V_T = \Phi$，　$S \in V_N$

$V = V_N \cup V_T$，称为文法G的文法符号集合

文法的定义

- 例2.1　文法G=（V_N，V_T，P，S）

  V_N = { S }，V_T ={ 0, 1 }

  P={ S→0S1，S→01 }

  S为开始符号

文法的定义

- 习惯上只将产生式写出。并有如下约定：
  - 第一条产生式的左部是开始符号
  - 用尖括号括起的是非终结符，否则为终结符。或者大写字母表示非终结符，小写字母表示终结符
  - G可写成G[S]，S是开始符号

G：S→aAb　A→ab　A→aAb　A→ ε

G[S]：A→ab　A→aAb　A→ ε　S→aSb

缩写形式 **G[S]：A→ab |aAb | ε　　S→aSb**

注意：元符号“|”读作“或”

例2.2　文法G=（V_N，V_T，P，S）

V_N ={标识符，字母，数字}

V_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>→z

<数字>→0, …,　<数字>→9　}

S=<标识符>

2.3.2 推导的形式定义

- 直接推导“⇒”

  $\alpha \to \beta$ 是文法G的产生式，若有v, w满足：

  $v=\gamma\ \alpha\ \delta$, $w=\ \gamma\ \beta\ \delta$，其中 $\gamma \in V^*$, $\delta \in V^*$

  则称v直接推导出w, 记作 $v \Rightarrow w$

  或w直接归约到v

例：G： **S→0S1， S→01**

S ⇒0S1 ⇒00S11 ⇒000S111 ⇒00001111

<程序>⇒<分程序>. ⇒ <变量说明部分> <语句>. ⇒... ⇒**VAR**<标识符>;**BEGIN READ(**<标识符>)**END.** ⇒ **VAR A;BEGIN READ(A) END.**

推导的定义

$\overset{+}{\Rightarrow}$

- 若存在$v \Rightarrow w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n = w, (n>0)$
- 则称$v \overset{+}{\Rightarrow} w$，v推导出w，或w归约到v。

$\overset{*}{\Rightarrow}$

- 若有$v \overset{+}{\Rightarrow} w$，或v=w，
- 则记为$v \overset{*}{\Rightarrow} w$

文法的句型、句子的定义

- 句型
  - 有文法G，若$S \overset{*}{\Rightarrow} x$，则称x是文法G的句型。
- 句子
  - 有文法G，若$S \overset{+}{\Rightarrow} x$，且$x \in V_T^*$，则称x是文法G的句子。

例：G： $S \rightarrow 0S1$， $S \rightarrow 01$

$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 00001111$

– 例：G[E]：E→E+T|T
T→T*F|F
F→(E)|a

E⇒E+T ⇒T+T ⇒F+T ⇒a+T ⇒a+T*F
⇒a+F*F ⇒a+a*F ⇒a+a*a

表示一切能用符号a，+，*，(和)构成的算术表达式

2.3.3 语言的形式定义

- 由文法G生成的语言记为L(G),它是文法G的一切句子的集合:
  $L(G)=\{x|S \overset{+}{\Rightarrow} x$，其中S为文法的开始符号，且$x \in V_T^*\}$

例：G： S→0S1， S→01

$L(G)=\{0^n1^n|n\geqslant 1\}$

- 例2.3 文法G[S]：

（1）S→aSBE

（2）S→aBE

（3）EB→BE

（4）aB→ab

（5）bB→bb

（6）bE→be

（7）eE→ee

$L(G)=\{ a^n b^n e^n \mid n \geqslant 1 \}$

S ⇒a S BE (S→aSBE)

⇒a aBEBE (S→aBE)

⇒aabEBE (aB→ab)

⇒aabBEE (EB→BE)

⇒aabbEE （bB→bb)

⇒aabbeE （bE→be)

⇒aabbee （eE→ee)

- G生成的每个串都在L(G)中
  L(G)中的每个串确实能被G生成

文法的等价

- 若L（G_1）=L（G_2），则称文法G_1和G_2是等价的。

如文法G_1[A]：A→0R 与G_2[S]：S→0S1 等价

A→01　　S→01

R→A1

练习：

- 已知语言描述，写出文法
  - 例：若语言由0、1符号串组成，串中0和1的个数相同，构造其文法。
  - A → 0B|1C
    B → 1|1A|0BB
    C → 0|0A|1CC
- 已知文法，写出语言描述
  - 例：G[E]：E→E+T|T
    T→T*F|F
    F→(E)|a

2.4 文法的类型

- **Chomsky将文法分为四种类型：**
  - **0型文法（短语文法）：** $G=(V_N,V_T,P,S)$对任一产生式$\alpha \rightarrow \beta$，都有$\alpha \in (V_N \cup V_T)^*$，且至少含有一个非终结符，$\beta \in (V_N \cup V_T)^*$
  - 1型文法（上下文有关文法）：对任一产生式$\alpha \rightarrow \beta$，都有$|\beta| \geqslant |\alpha|$，仅仅 $S \rightarrow \varepsilon$ 除外
  - 1型文法也可描述为$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$
  - 2型文法（上下文无关文法）：对任一产生式$\alpha \rightarrow \beta$，都有$\alpha \in V_N$，$\beta \in (V_N \cup V_T)^*$
  - 3型文法（正规文法）：任一产生式$\alpha \rightarrow \beta$的形式都为$A \rightarrow aB$或$A \rightarrow a$，其中$A \in V_N$，$B \in V_N$，$a \in V_T$

文法的类型

- 例：1型（上下文有关）文法

  文法G[S]：　S→aSBE

  S→aBE

  EB→BE

  aB→ab

  bB→bb

  bE→be

  eE→ee

文法的类型

- 例：1型（上下文有关）文法

  文法G[S]：

  S→CD

  C→aCA

  C→bCB

  AD→aD

  BD→bD

  Aa→bD

  Ab→bA

  Ba→aB

  Bb→bB

  C→ ε

  D→ ε

$L(G)=\{ww \mid w \in \{a, b\}^*\}$

文法的类型

- 例：2型（上下文无关）文法

文法G[S]：　S→aB|bA

A→a|aS|bAA

B→b|bS|aBB

文法G[S]：　S→0A|1B|0

A→0A|1B|0S

B→1B|1|0

文法的类型

- 例：定义标识符的3型（正规）文法

文法**G[I]**：

I → i**T**

I → i

T → i**T**

T → d**T**

T → i

T → d

文法和语言

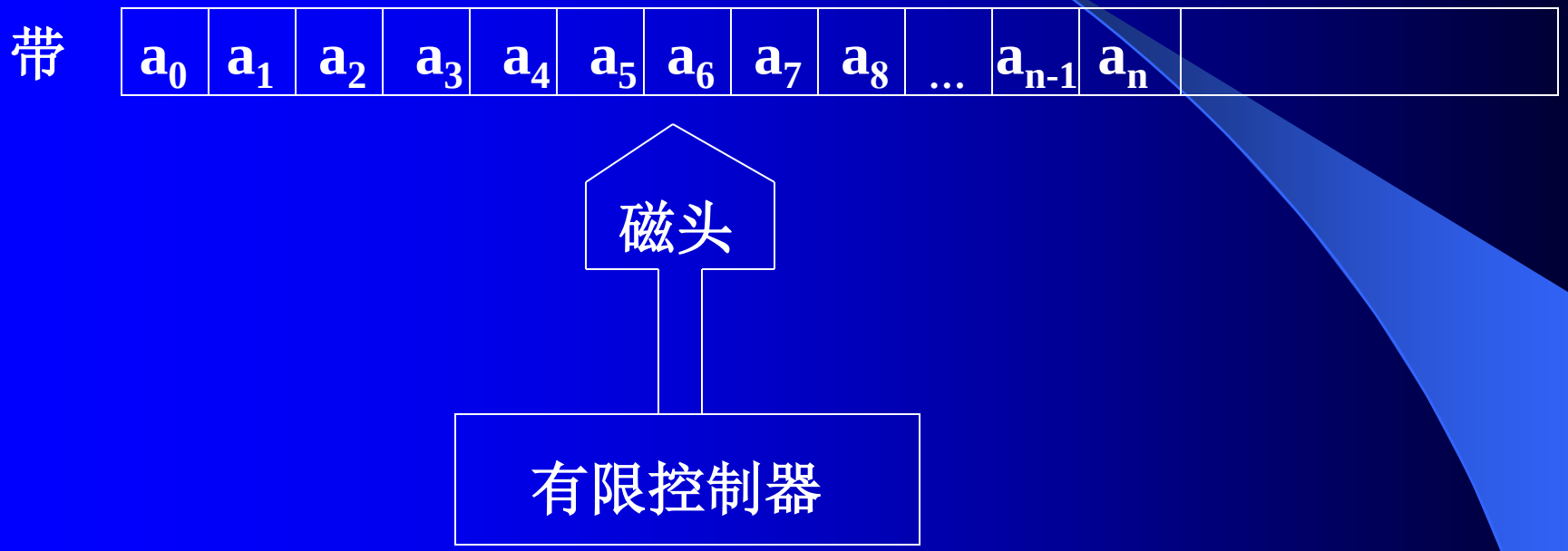
- 0型文法产生的语言称为0型语言
- 1型文法或上下文有关文法（ CSG ）产生的语言称为1型语言或上下文有关语言（CSL）
- 2型文法或上下文无关文法（ CFG ）产生的语言称为2型语言或上下文无关语言（ CF L ）
- 3型文法或正则（正规）文法（ RG ）产生的语言称为3型语言正则（正规）语言（ RL ）

文法和识别系统

- 0型文法（短语文法）的能力相当于图灵机，可以表征任何递归可枚举集，而且任何0型语言都是递归可枚举的

- 1型文法（上下文有关文法）：产生式的形式为$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$，即只有A出现在$\alpha_1$和$\alpha_2$的上下文中时，才允许$\beta$取代A。其识别系统是线性界限自动机。

图灵机



文法的类型

- **2**型文法（上下文无关文法、**CFG**）：产生式的形式为A→β， β取代A时与A的上下文无关。其识别系统是不确定的下推自动机。
- **3**型文法（正规文法、右线性文法）：产生的语言是有穷自动机（**FA**）所接受的集合

2.5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
  - 算术表达式
  - 语句
    - 赋值语句
    - 条件语句
    - 读语句
    - ……

例：算术表达式上下文无关文法表示

- 文法G=({E}, {+,*,I,(,)}, P, E}

  P: E → i

  E → E+E

  E → E*E

  E → (E)

例：条件语句上下文无关文法表示

<条件语句>→if<条件>then<语句>
| if<条件>then<语句>else <语句>

2.5.1上下文无关文法的语法树

- 用于描述上下文无关文法的*句型推导*的直观方法

例: G[S]:

S→aAS

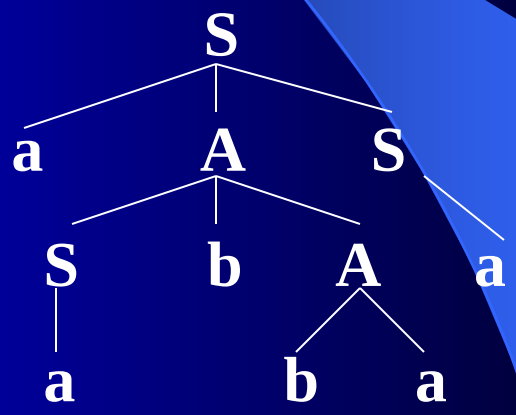
A→SbA

A→SS

S→a

A→ba

句型aabbaa的语法树（推导树）



叶子结点：树中没有子孙的结点。
从左到右读出推导树的叶子标记，所得的句型为推导树的结果。也把该推导树称为该句型的语法树。

上下文无关文法的语法树

- 给定文法G，对于G的任何句型都能构造与之关联的语法树（推导树）。这棵树满足下列4个条件：
  1、每个结点都有一个V中的符号作标记
  2、根的标记是开始符号S
  3、若一结点n至少有一个它自己除外的子孙，并且n有标记A，则$A \in V_N$
  4、如果结点n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$A \rightarrow A_1A_2,\ldots,A_k$一定是P中的一个产生式

上下文无关文法的语法树

- 推导过程中施用产生式的*顺序*

例: G[S]:

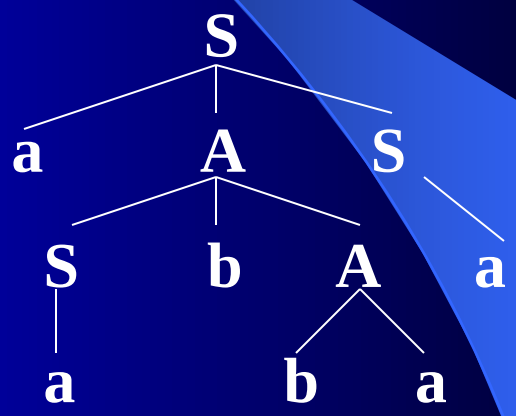
S→aAS

A→SbA

A→SS

S→a

A→ba



S⇒aAS⇒aAa⇒aSbAa⇒aSbbaa⇒aabbaa

S⇒aAS⇒aSbAS⇒aabAS⇒aabbaS⇒aabbaa

S⇒aAS⇒aSbAS⇒aSbAa⇒aabAa⇒aabbaa

- 最左（最右）推导：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）非终结符进行替换
- 最右推导被称为规范推导。
- 由规范推导所得的句型称为规范句型

2.5.2 二义性

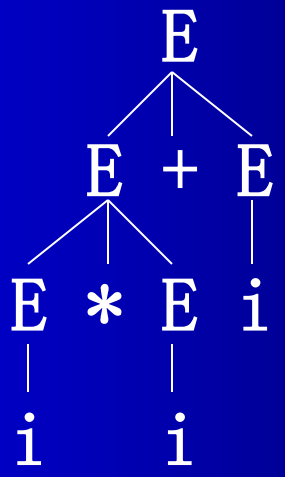
- 问题：一个句型是否对应唯一的一棵语法树？
- 例：G[E]：

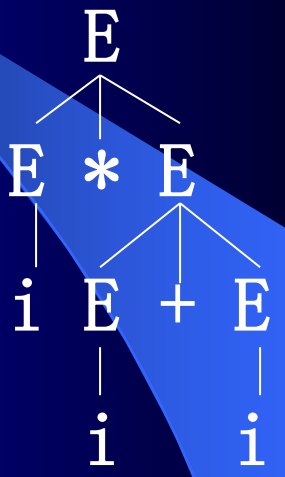
  E → i

  E → E+E

  E → E*E

  E → (E)





句型 i*i+i 的两个不同的最左推导：

推导1：$E \Rightarrow E+E \Rightarrow E*E+E \Rightarrow i*E+E \Rightarrow i*i+E \Rightarrow i*i+i$

推导2：$E \Rightarrow E*E \Rightarrow i*E \Rightarrow i*E+E \Rightarrow i*i+E \Rightarrow i*i+i$

二义文法

- 若一个文法存在某个句子对应两棵不同的语法树，则称这个文法是*二义*的。
  或者，若一个文法存在某个句子有两个不同的最左（右）推导，则称这个文法是*二义*的。
- 产生某上下文无关语言的每一个文法都是二义的，则称此语言是*先天二义*的。

二义文法

- 判定任给的一个上下文无关文法是否二义，或它是否产生一个先天二义的上下文无关语言，这两个问题是递归不可解的。但可以为无二义性寻找一组充分条件（不一定是必要的）。
- 二义文法改造为无二义文法

G[E]: E → i
E → E+E
E → E*E
E → (E)

G[E]: E → T|E+T
T → F|T*F
F → （E）|i

规定优先顺序和结合律

2.6句型的分析

- *句型分析* 就是识别一个符号串是否为某文法的句型，是某个推导的构造过程。
- 在语言的编译实现中，把完成句型分析的程序称为*分析程序*或*识别程序*。分析算法又称*识别算法*。
- *从左到右的分析算法*，即总是从左到右地识别输入符号串，首先识别符号串中的最左符号，进而依次识别右边的一个符号。

句型的分析

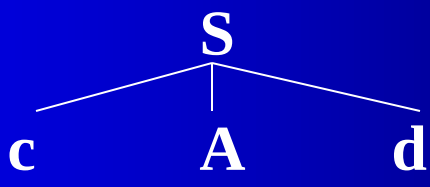
- 分析算法可分为：
- *自上而下分析法*：
  从文法的开始符号出发，反复使用各种产生式，寻找与输入符号匹配的推导。
- *自下而上分析法*：
  从输入符号串开始，逐步进行归约，直至归约到文法的开始符号。
- 两种方法反映了两种不同的语法树的构造过程

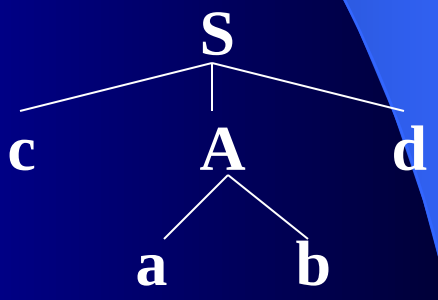
自上而下的语法分析

- 例：文法G：S → cAd
  A → ab
  A → a

识别输入串w=cabd是否该文法的句子

S



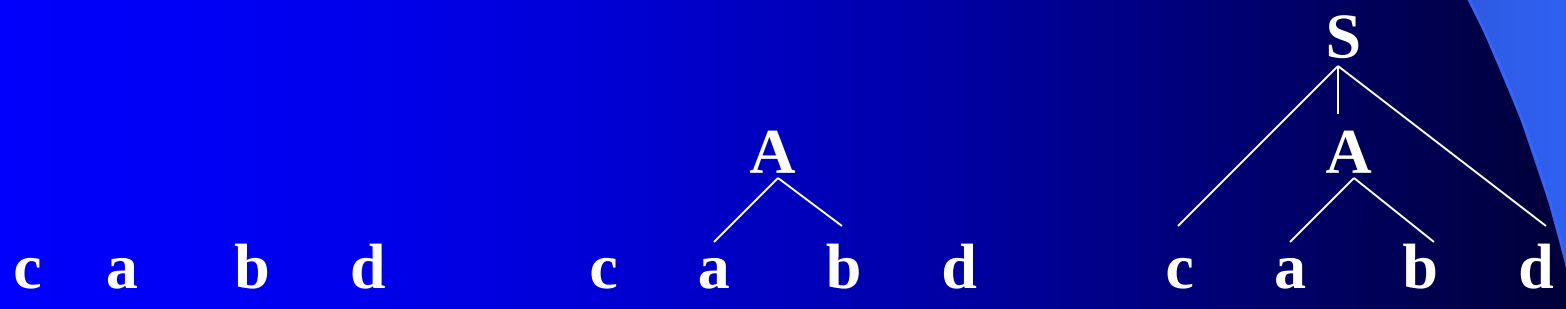


推导过程：S ⇒ cAd ⇒ cabd

自下而上的语法分析

- 例：文法G：S → cAd
  A → ab
  A → a

  识别输入串w=cabd是否该文法的句子



归约过程：S ⇒ cAd ⇒ cabd

句型分析的有关问题

1）如何选择使用哪个产生式进行推导？

假定要被代换的最左非终结符号是V，且有n条规则：V→A1|A2|…|An，那么如何确定用哪个右部去替代V？

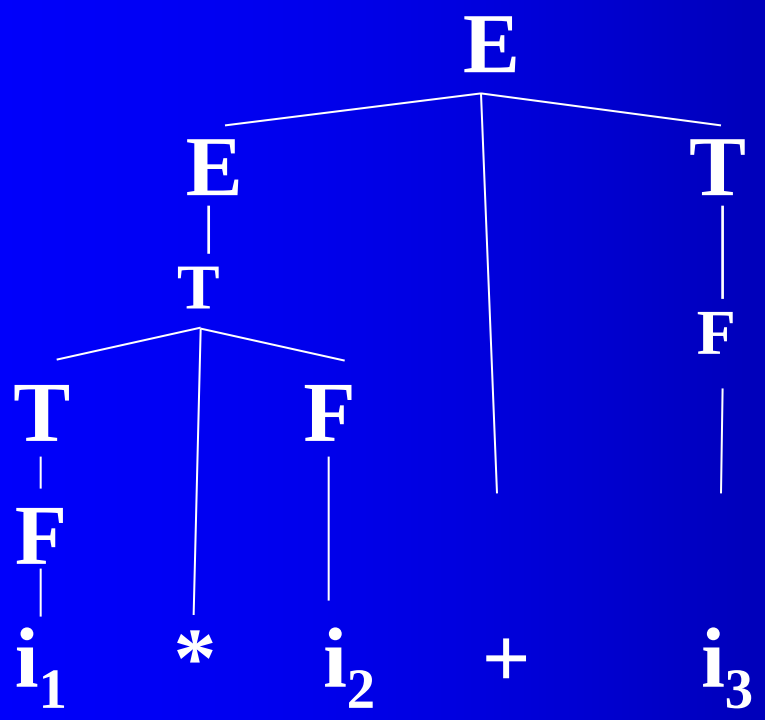
2）如何识别可归约的串？

在自下而上的分析方法中，在分析程序工作的每一步，都是从当前串中选择一个子串，将它归约到某个非终结符号，该子串称为“可归约串”

句型分析

- 短语、直接短语、句柄的定义：
- 文法G[S]，$S \overset{*}{\Rightarrow} \alpha A \delta$ 且$A \overset{+}{\Rightarrow} b$则称b是句型 α b δ 相对于非终结符A的短语。若有$A \Rightarrow b$则称b是句型 α b δ 相对于该规则$A \rightarrow b$的直接短语。一个句型的最左直接短语称为该句型的句柄。

句型分析



G[E]：E→E+T|T
T→T*F|F
F→(E)|i

句型：i*i+i

短语：

直接短语：

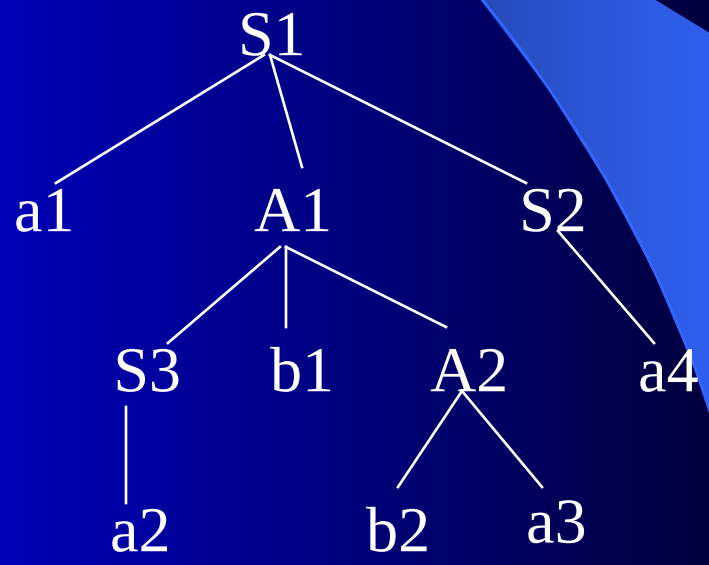
句柄：

子树与短语、句柄

- 子树：在一棵语法树中，由某一结点及其所有分支组成的部分称为原树的一棵子树。
- 一棵子树的所有叶子自左至右排列起来形成一个相对子树根的短语。（如上例）
- 一个句型的句柄是这个句型的分析树中最左那棵只有父子两代的子树的所有叶子的自左至右排列。

- 例：设文法G[S]：
- S →aAS|a　　A →SbA|SS|ba
- 则aabbaa是该文法的一个句子，求此句子的短语和句柄

短语：
1、a1a2b1b2a3a4
2、a2b1b2a3
3、a4
4、a2
5、b2a3
句柄：
a2



2.7有关文法实用中的一些说明

- 有关文法的实用限制
- 上下文无关文法中的 ε 规则

有关文法的实用限制

- 文法中不得含有*有害规则*和*多余规则*
  - 有害规则：形如U→U的产生式。会引起文法的二义性
  - 多余规则：指文法中任何句子的推导都不会用到的规则
    - 1）文法中某些非终结符不在任何规则的右部出现，该非终结符称为不可到达
    - 2）文法中某些非终结符，由它不能推出终结符号串来，称为不可终止的

有关文法的实用限制

- 对于文法G[S]，为了保证任一非终结符A在句子推导中出现，必须满足如下两个条件：

  1）A必须在某句型中出现。
  2）必须能从A推出终结符号串t来。

- 即1）文法G[S]，对某两个符号串 α 和 δ：
  $S \overset{*}{\Rightarrow} \alpha A \delta$　2）$A \overset{+}{\Rightarrow} t$，$t \in V_T^*$

化简文法

- 例：G[S]

  1) S→Be　　S→Be

  *2) B→Ce　　B→Af

  3) B→Af　　A→Ae

  4) A→Ae　　A→e

  5) A→e

  *6) C→Cf（不可终止的）

  *7) D→f （不可到达的 ）

上下文无关文法中的ε规则

- 具有形式$A \rightarrow \varepsilon$的规则称为ε规则，其中$A \in V_N$
- 某些著作和讲义中限制这种规则的出现。因为ε规则会使有关文法的一些讨论和证明变得复杂

总结

- 文法的直观概念
- 符号、符号串及其集合的运算
- 文法和语言的形式定义
- 文法的类型
- 上下文无关文法及其语法树
- 句型的分析
- 文法实用中的一些说明