第 2 章　形式语言基础

计算机处理语言，首先应考虑语言的形式化、规范化，使其具有可计算性和可操作性，这就是形式语言理论研究的问题。

形式语言诞生于 1956 年，由 chomsky 创立。语言研究涉及三个方面：语法、语义和语用；我们侧重语法的研究。

⁎ 形式语言的基本观点是：语言是符号串之集合！（语言就是一句一句话）

⁎ 形式语言理论研究的基本问题是：研究符号串集合的表示方法、结构特性以及运算规律。

2.1　形式语言是符号串集合

定义 2.1 形式语言：字母表上的符号，按一定的规则组成的所有符号串集合。其中的每个符号串称为句子。

定义 2.2 字母表：元素（符号）的非空有限集合；（构成字符串的最基本符号不能再切割了）。

定义 2.3 符号串：符号的有限序列。

在讨论符号串时，注意是基于某个字母表的，例如对于 0，1 构成的字母表，其上的符号串可以是 010，0101110，……

定义 2.4 符号串集合：有限个或者无限个符号串组成的集合。

定义 2.5 规则：以某种形式表达的在一定范围内共同遵守的规章和制度。这里指符号串的组成规则。

形式语言概念示例：

例 2.1 $L_1 = \{00, 01, 10, 11\}$;

字母表有 $\sum_1 = \{0, 1\}$;

句子有：00,01,10,11

例 2.2 $L_2 = \{ab^m c, b^n \mid m > 0, n \geqslant 0\}$;

字母表有 $\sum_2 = \{a, b, c\}$;

句型 1：$ab^m c$,

有句子：$abc, abbc, abbbc, \ldots$

句型 2：b^n,

有句子：$\varepsilon, b, bb, bbb, \ldots$

注意：

1. 句型不代表某一个具体的串，代表一类串；

2. m 是大于 0 的，n 是大于等于 0 的；

3. $b^0 = \varepsilon$(空符号串), $b^1 = b, b^2 = bb, b^3 = bbb, \ldots$;

4. L_1 为有限语言，L_2 为无限语言, 包括无限多个符号串。

2.1.1 符号串（集合）的运算

符号串的运算：

定义 2.6 连接：$\alpha \cdot \beta = \alpha\beta$ 如 $a \cdot b = ab$

连接：设有 x, y 两个符号串，它们的连接操作是把它们连在一起，结果是 xy。举个例子，$x = ab$，$y = c$，则连接后的符号串为 $xy = abc$。

特例：如果一个字符串和一个空字符串连接的话，其结果为自己本身，即：$x\varepsilon = \varepsilon x = x$

定义 2.7 或：$\alpha \mid \beta = \alpha$（或者 β）

定义 2.8 方幂：$\alpha^n = \prod^{n} \alpha = \alpha\alpha^{n-1} = \alpha^{n-1}\alpha$

$\alpha^0 = \varepsilon$(空符号串)

$\alpha^1 = \alpha; \alpha^2 = \alpha\alpha; \ldots$

方幂：设 x 是一个符号串，则 x 的方幂 $x^2 = xx, x^3 = x^2x, \ldots, x^n = x^{n-1}x$。

特例：$x^1 = x, x^0 = \varepsilon$

定义 2.9 闭包：α 的正闭包：$\alpha^+ = \alpha^1 \mid \alpha^2 \mid \ldots \mid \alpha^n \mid \ldots$

α 的星闭包：$\alpha^* = \alpha^0 \mid \alpha^1 \mid \alpha^2 \mid \ldots \mid \alpha^n \mid \ldots$

闭包：设 x 是一个符号串，则 x 的闭包 $x^* = x^0 \mid x^1 \mid x^2 \mid x^3 \mid \ldots \mid x^n \mid \ldots$ 其中：‘|’是元符号（描述符号），意义为：或者是，即：x^* 是一种泛指。

⁎ 符号串运算示例

例 2.3

1. $abc \cdot de = abcde$

2. $ab \mid cd = ab$(或者 cd)

3. $(a \mid b)^1 = (a \mid b) = a \mid b$

$(a \mid b)^2 = (a \mid b)(a \mid b) = aa \mid ab \mid ba \mid bb$

$(a \mid b)^* = (a \mid b)^0 \mid (a \mid b)^1 \mid (a \mid b)^2 \mid \ldots$

即：$(a \mid b)^* = (a \mid b)^n, n \geqslant 0$

a 或 b 的星闭包表示由 a 或 b 所构成的串（包含空串）, 如果不包含空串则写成下面的形式。

$$(a \mid b)^+ = (a \mid b)^n, n > 0$$

符号串集合的运算

设 A 和 B 为两个符号串集合，则：

定义 2.10 乘积：AB = {xy | x ∈ A且y ∈ B}

例如：$U = \{a, b\} V = \{c\}$则$UV = \{ac, bc\}$

特例：$\{\varepsilon\} \cdot U = U \cdot \{\varepsilon\} = U\varnothing \cdot U = U \cdot \varnothing = \varnothing$

定义 2.11 和：$A \cup B = A + B = \{x \mid x \in A$或$x \in B\}$

找到 A 或者 B 中的元素取并集

定义 2.12 方幂：$A^n = \prod_{n} A = AA^{n-1} = A^{n-1}A$

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

$A^1 = A; A^2 = AA; A^3 = AAA; \ldots$

方幂是若干个集合进行乘积，所谓方幂就是 $x^n = x^{n-1}x$。

特例：$x^1 = x, x^0 = \varepsilon$

定义 2.13 闭包：A 的正闭包：$A^+ = A^1 \cup A^2 \cup \ldots \cup A^n \cup \ldots$

A 的星闭包：$A^* = A^0 \cup A^1 \cup A^2 \cup \ldots \cup A^n \cup \ldots$

把 A 的一次幂到 n 次幂求并集叫做集合 A 的正闭包，如果考虑 A 的 0 次幂，包含空串的集合称为 A 的星闭包。

符号串集合运算示例：

例 2.4 设 $A = \{a, b\}, B = \{c, d\}$

则 $A + B = \{a, b, c, d\}$

则 $AB = \{xy \mid x \in A, y \in B\} = \{ac, ad, bc, bd\}$

例 2.5 设 $A = \{a\}$

则 $A^* = A^0 \cup A^1 \cup A^2 \cup \ldots \cup A^n \cup \ldots$

$= \{\varepsilon\} + \{a\} + \{aa\} + \{aaa\} + \ldots$

$= \{\varepsilon, a, aa, aaa, \ldots\}$

$= \{a^n \mid n \geqslant 0\}$

假设 A 只包含一个符号串 a，通过并集运算得到一个包含空串和只由 a 构成的符号串的集合，它可以看作字母 a 的一个语言。

我们把 A 扩展为包含 a 和 b 的集合，那 A 的星闭包是什么呢？

例 2.6 设 $A = \{A, B\}, A^* = ?$

$\because A^* = A^0 \cup A^1 \cup A^2 \cup \ldots \cup A^n \cup \ldots$

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

$A^1 = A = \{a, b\}$

$A^2 = A \cdot A = \{a, b\} \cdot \{a, b\} = \{aa, ab, ba, bb\}$

$A^3 = A \cdot A^2 = \{a, b\} \cdot \{aa, ab, ba, bb\} = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

由上述分析可知，A 的星闭包由 a 和 b 构成的所有符号串所组成的集合，包括空串。

推论：若 A 为任一字母表，则 A* 就是该字母表上的所有符号串（包括空串）的集合。

如果把 a 和 b 看成字母表中的两个元素，A 的星闭包就是以 a 和 b 这两个符号构成的语言，当然这里面是包含空串的。A 的星闭包或者 A 的加闭包实际上是在定义语言，给一个字母表，这个操作就可以把字母表所对应的所有符号串都生成出来。

2.1.2 符号串集合的文法描述

前面已经讨论了语言就是符号串的集合，也介绍了集合的基本操作。但是如果给你一个语言，不管是机器语言还是 C++ 语言，我们如何去描述他呢？这是一个比较深刻的问题。我们需要一种能刻画客观世界规律性的描述方式。从我们计算机现在的手段来讲，一般用一些可数学化描述的方式把客观事物描述出来。比如算法或者数据结构，包括前面讲的形式语言自动机，归根结底还是一种数学化的描述。我们现在说符号串的集合表示一种语言，那么我们如何去描述这种语言呢？其中一种非常重要的工具就叫文法。这个概念很重要！计算机中很多问题都可以用文法来描述，甚至很多你用的事务都可以用文法来描述，比如统计符号串，脚本语言中正则表达式的匹配，实际上都是一种文法的体现。那么什么是文法呢？

例 2.7 $L = \{ab^n c \mid n \geqslant 0\}$

字母表：$\sum = \{a, b, c\}$

展开：$L = \{ac, abc, abbc, abbbc, \ldots\}$

假如我们现在有这样一个语言，这个语言包含字符串的形式是 a、b 的 n 次幂和 c 的乘积（$n \geqslant 0$），他表示的是字母表上 abc 这三个字母所构成的一个语言，这个语言所包括的符号串模式是，一个 a 和一个 c，中间夹了零个或若干个 b，对于这样一个语言如果我们用文法来描述他的话，应该是什么样子呢？我们用如下几条文法规则来描述上面语言，文法规则也叫产生式规则，这些规则就构成了上述语言生成的规则。

文法规则

$$S \to aAc$$

$$A \to bA \mid \varepsilon$$

其中 S 和 A 可以理解为一些抽象的变量，如果把 S 看成句子的话，A 就是句子里的一个短语，后面将 S 和 A 定义为非终结符；a、b、c 是字母表中的字母，这里用 ε 表示空符号串，箭头表示推导或产生的意思，左部产生右部，$\mid$ 表示或的关系。第二组规则也可以写成两条规则：$A \to bA, A \to \varepsilon$。

怎样利用上述文法规则表示语言 L?

从开始符号出发，对符号串中的定义对象（非终结符），采用推导的方法（用其规则右部替换左部）产生新的符号串，如此进行，直到新符号串中不再出现定义的对象为止，则最终的符号串就是一个句子。

给定下述文法规则，我们分析其所定义的语言：

文法规则

$$S \to aAc$$

$$A \to bA \mid \varepsilon$$

句子产生过程（$\Rightarrow$ 推导算符）：

1. $S \Rightarrow aAc \Rightarrow a\varepsilon c = ac$
2. $S \Rightarrow aAc \Rightarrow abAc \Rightarrow ab\varepsilon c = abc$
3. $S \Rightarrow aAc \Rightarrow abAc \Rightarrow abbAc \Rightarrow abbc$

   …

$\therefore S \overset{+}{\Rightarrow} ab^n c, n \geqslant 0$

(1) 假定 S 是起始符号，S 推出 aAc，再把 A 替换为 ε，得到了 $a\varepsilon c$，最终得到了 ac

(2) 假定 S 是起始符号，S 推出 aAc，把 A 替换为 bA，再把 A 替换为 ε，得到了 $ab\varepsilon c$，最终得到了 abc

(3) 假定 S 是起始符号，S 推出 aAc，把 A 替换为 bA，把 A 替换为 bA，再把 A 替换为 ε，得到了 $abb\varepsilon c$，最终得到了 $abbc$

以此类推，可以得到 $S \overset{+}{\Rightarrow} ab^n c, n \geqslant 0$，$\Rightarrow$ 表示只经过一次推导，$+$ 表示至少经过一次推导，$\overset{+}{\Rightarrow}$ 代表连续推导。

2.2 形式语言由文法定义的

2.2.1 什么是文法？

我们这门课重点讨论上下文无关文法，上下文无关文法可以理解为规则均是由一个非终结符构成的左部推导出右部，这样的文法都可以称之为上下文无关文法。

定义 2.14 文法（grammar）是规则的有限集，其中的上下文无关文法可以定义为四元组：$\mathrm{G(Z)} = (\mathrm{V_N}, \mathrm{V_T}, \mathrm{Z}, \mathrm{P})$

$\mathrm{V_N}$: 非终结符集（定义的对象集，如：语法成分等)；后面语法树中的中间节点都是非终结符，从计算的角度看，可以把非终结符看作变量，可以被替换的

$\mathrm{V_T}$: 终结符集（字母表）

Z : 开始符号（研究范畴中，最大的定义对象)；整个推导开始的状态

P : 规则集（又称产生式集)；规则集包含若干个产生式，每个产生式左部由单一的非终结符构成，右部是由终结符和非终结符构成的任意符号串，形如:$\mathrm{A} \to \alpha$ 或者 $\mathrm{A} \to \alpha \mid \beta$。

其中，描述符号：$\to$（定义为)，$\mid$（或者是)，文法符号：$\mathrm{Z, A} \in \mathrm{V_N}, \alpha, \beta \in (\mathrm{V_N} + \mathrm{V_T})^*$

Z 和 A 都属于非终结符集；$\mathrm{V_N} + \mathrm{V_T}$ 表示这个集合中的元素要么是终结符要么是非终结符，$\mathrm{V_N} + \mathrm{V_T}$ 的星闭包代表以这个集合里面元素构成的所有符号串，即把 $\mathrm{V_N} + \mathrm{V_T}$ 集合作为字母表所构成的所有符号串。

2.2.2 文法是怎样定义语言的？

定义 2.15 设有文法 G(Z)，L(G) 为 G 所定义的语言，则 $\mathrm{L(G)} = \left\{ \mathrm{x} \mid \mathrm{Z} \overset{+}{=} \mathrm{x}, \mathrm{x} \in \mathrm{V_T^*} \right\}$

有了文法我们如何定义语言呢？一个语言就是由这个文法所生成的字符串构成的。$G(Z)$ 是一个文法，$L(G)$ 为 $G(Z)$ 所定义的语言，他是一个由若干个元素构成的集合，元素是从文法其实符号 Z 经过加推导出的由终结符构成的符号串 x。这个定义包含两个层面，第一，起始符是文法开始符号 Z，推导一定是由 Z 开始的；第二，语言中的符号串 x 一定是终结符串，不能含有非终结符，如果含有非终结符还要继续推导，直到这个串中没有非终结符，这样才能形成最终的语言。

即：一个文法所定义的语言，是由该文法开始符号推导出的所有仅含终结符的符号串之集合。其中的每个符号串皆称为句子。

定义 2.16 标识符: 字母开头的字母、数字序列。

例 2.8 标识符的文法。

标识符在我们书写程序的时候会使用，声明一个变量，或者写一个函数就会用到标识符，标识符是不可以以数字开头的。现在标识符的定义也不仅仅包含字母和数字，下划线也可以作为标识符的一部分。

标识符文法的非终结符包括两个部分：I(标识符)，A(标识符尾)

终结符包括两个部分：ℓ(字母)，d(数字）

起始符：I

规则集：$\mathrm{I} \to \ell\mathrm{A} \mid \ell$

$\mathrm{A} \to \ell\mathrm{A} \mid \mathrm{dA} \mid \varepsilon$

不难看出，上述文法定义的符号串是一个以字母开头的字母和数字构成的序列。

例 2.9 无符号整数文法

能否写出一个无符号整数的文法？可以写成这样的形式 $N \to dN \mid d$，其中 d 表示 0–9 的数字。这个文法很特殊，N 既出现在了文法的左部又出现在了文法的右部，这就构成了递归的形式，递归是描述无限语言非常有效的方式。无符号整数的元素非常多，用这种方式就可以描述无限个无符号的整数。

上述无符号整数文法规则是否存在问题？请读者自行分析。

⁎ 标识符文法所定义的语言求解：

标识符文法是如何表达标识符的呢？如果想看这个文法表示的是什么内容，那要知道这个文法生成的句子 (符号串) 是什么样子。单独从文法规则是看不出来的，我们需要进行一些简单的运算。

上面构造的标识符文法属于正规文法 (定义在后) 类，正确性检验较容易；下面给出一个算法：

$$\left.\begin{array}{l} \text{令：} I = \ell A \mid \ell \\ \qquad A = \ell A \mid dA \mid \varepsilon \end{array}\right\} \Rightarrow \text{正规方程式}$$

⁎ 求解 I 值步骤：

这个等式不是代数上的等式，而是符号串的替换：I 可以用 ℓA 或 ℓ 替换。我们要求解 I 是什么，从起始符号 A 开始，它可以得到 ℓA 或 dA 或 ε，那 A 是什么呢？

$A = (\ell \mid d)A, A = (\ell \mid d)^2 A, \ldots, A = (\ell \mid d)^n A$。$A = \ell A \mid dA$ 对应 $A = (\ell \mid d)A$，这个等式左边右边都有变量，那右边的变量我们可以用同样的 A 进行替换。我们就可以生成 $A = (\ell \mid d)^2 A$，同样也可以生成 $A = (\ell \mid d)^n A$，只有当 A 推出 ε 时停止，已经全是终结符串了，最终得到 $A = (\ell \mid d)^n A, n \geqslant 0$，当 $n = 0$ 时包含了 A 推出 ε 的情况。再把 A 代入 $I = \ell A \mid \ell$ 中得到了 $I = \ell(\ell \mid d)^n, n \geqslant 0$，这是标识符串，起始是字母，后面是由字母和数字构成的符号串。

例 2.10 简单算术表达式文法（这个文法务必记牢）

这个文法的非终结符包含 E(算术表达式)、T(项)、F(因式)

终结符包含 i(变量或常数), $+, -, *, /, (,)$

起始符是 E，由 E 开始推导

规则

规则：$E \to T \mid E + T \mid E - T$

$T \to F \mid T * F \mid T/F$

$F \to i \mid (E)$

此文法定义了算术表达式的层次嵌套结构：

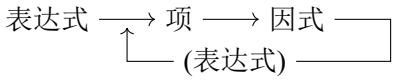


图 2.1: 算术表达式的层次嵌套结构

定义了优先级，在推导的时候先推 $+$、$-$，后推 $*$、$/$，从运算的角度，因式先做，实际上对应的是规约的过程，当然这个因式可以包括表达式，可以无限嵌套下去。

算术表达式文法应用示例：

证明 $\mathrm{i}*(\mathrm{i}+\mathrm{i}-\mathrm{i})$ 是文法 $G(E)$ 的一个句子 (即合法的算术表达式):

根据语言定义式2.15，合法的算术表达式是指：

$\mathrm{E} \overset{+}{\Rightarrow} \mathrm{i}*(\mathrm{i}+\mathrm{i}-\mathrm{i})$ 成立吗？

首先 i 可能代表的是一个数字、标识符、变量，不管 i 代表的是什么，我们把 i 当成一个终结符来看，我们想证明这个符号串是否是表达式文法定义的句子。

语言由起始符开始经过若干次推导得到的终结符串，这样的串构成的集合称之为语言，反过来说，如果我们要证明这个串是语言中的一个句子，那就看能不能从文法起始符经过加推导得到这个符号串。

$\mathrm{E} \Rightarrow \mathrm{T} \Rightarrow \mathrm{T}^*\,\mathrm{F} \Rightarrow \mathrm{T}^*(\mathrm{E}) \Rightarrow \mathrm{T}^*(\mathrm{E}-\mathrm{T}) \Rightarrow \mathrm{T}^*(\mathrm{E}+\mathrm{T}-\mathrm{T}) \Rightarrow \mathrm{i}^*(\mathrm{E}+\mathrm{T}-\mathrm{T}) \Rightarrow \ldots \Rightarrow \mathrm{i}^*(\mathrm{i}+\mathrm{i}-\mathrm{i})$

经过上述推导，可以看出符号串 i*(i+i-i) 是表达式文法定义的句子。

$\therefore \mathrm{E} \overset{+}{\Rightarrow} \mathrm{i}*(\mathrm{i}+\mathrm{i}-\mathrm{i})$

2.3 主要语法成分的定义

2.3.1 文法的运算问题

文法有一些基本的运算。文法有两种基本运算：推导，规约。

设 $\mathrm{x},\mathrm{y} \in (\mathrm{V_N}+\mathrm{V_T})^*, \mathrm{A} \to \alpha \in \mathrm{P}$

假设 x 和 y 是由终结符和非终结符构成的符号串，$\mathrm{A} \to \alpha$ 是一条规则。

定义 2.17 直接推导 ($\Rightarrow$)：$\mathrm{xAy} \Rightarrow \mathrm{x}\alpha\mathrm{y}$

这个过程称之为使用上述规则完成的一个推导，将符号串中的一个非终结符 A 替换成所对应规则的右部 α，这个过程称之为直接推导。$\Rightarrow$ 称为直接推导算符。

即：指用产生式的右部符号串替换左部非终结符。

定义 2.18 加推导 ($\overset{+}{\Rightarrow}$)：一个符号串经过上述直接推导若干次得到的一个新的串，简记为 $\alpha \overset{+}{\Rightarrow} \beta$，$\overset{+}{\Rightarrow}$ 称为加推导算符。加推导需要至少一次推导。

即：指一步或一步以上的直接推导运算。

定义 2.19 星推导 ($\overset{*}{\Rightarrow}$)，α 经过若干次推导得到 β，也可以一步都不推导。α 可以星推导得到 α，但是 α 不能加推导得到 α。

即：指零步或零步以上的直接推导运算。

定义 2.20 直接规约 ($\dot{\Rightarrow}$)：$\mathrm{x}\alpha\mathrm{y} \dot{\Rightarrow} \mathrm{xAy}$

推导对应的是自上而下的分析，归约对应自下而上的分析。可以把规约看作是推导的逆运算，$\dot{\Rightarrow}$ 称之为

直接归约算符。

即：直接归约是直接推导的逆运算，用产生式的左部非终结符替换右部符号串。

定义 2.21 加归约 ($\overset{+}{\Rightarrow}$) 经过若干次归约，若干次左部替换右部的操作，加归约规定至少归约一次，$\overset{+}{\Rightarrow}$ 为加归约算符。

即：指一步或一步以上的直接归约运算。

定义 2.22 星归约 ($\overset{*}{\Rightarrow}$)，若干次的直接归约，可以是零次。

即：指零步或零步以上的直接归约运算。

定义 2.23 最左推导 ($\overset{+}{\underset{\ell}{\Rightarrow}}$)——每次推导皆最左非终结符优先。

定义 2.24 最左归约 ($\overset{+}{\underset{\ell}{\Rightarrow}}$)——每次规约皆最左可归约串优先。

为什么会定义最左推导和最左归约，最左推导和一般的推到相比有什么好处？最左归约和一般的归约相比有什么好处？读者可以思考一下。

例 2.11 给定一个符号串 $i+i*i$：

算数表达式文法

$$G(E):E\to T\mid E+T\mid E-TT\to F\mid T*F\mid T/FF\to i\mid (E)$$

最左推导（从开始符号出发）过程：

$$E\Rightarrow E+T\Rightarrow T+T\Rightarrow F+T\Rightarrow i+T\Rightarrow i+T*F\Rightarrow i+F*F\Rightarrow i+i*F\Rightarrow i+i*i$$

$$\therefore E\overset{+}{\underset{\ell}{\Rightarrow}} i+i*i$$

最左归约（从符号串出发）过程：

$$i+i*i\Rightarrow F+i*i\Rightarrow T+i*i\Rightarrow E+i*i\Rightarrow E+F*i\Rightarrow E+T*i\Rightarrow E+T*F\Rightarrow E+T\Rightarrow E$$

$$\therefore i+i*i\overset{+}{\underset{\ell}{\Rightarrow}} E$$

既然推导和归约是相逆的过程，那最左推导和最左归约是不是一个互逆的过程？实际上最左推导和最左归约并不互逆。

2.3.2 句型、句子和语法树

设有文法 $G(Z)=(V_N,V_T,Z,P)$，则：

定义 2.25 句型：若有 $Z\overset{+}{\Rightarrow}\alpha$，则 α 是句型

α 既可以包含终结符也可以包含非终结符。

定义 2.26 句子：若有 $Z\overset{+}{\Rightarrow}\alpha$，且 $\alpha\in V_T^*$，则 α 是句子。即句子是由开始符号加推导出的任一终结符号串。

句子必须都由终结符构成。

定义 2.27 语法树：句型（句子）产生过程的一种树结构表示。

树根——开始符号；树叶——给定的句型（句子）。

树对应了整个推导的过程，反过来说，在上下文无关文法的体系下，一个语法树就对应了一个推导，而且在一些限定条件下，他会唯一对应一个推导。

语法树的构造算法：

1. 置树根为开始符号；
2. 若采用了推导产生式：$A \to x_1x-2\ldots x_n$，则有子树：

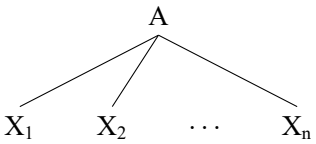


图 2.2: 推导产生式的子树

3. 重复步骤 (2)，直到再没有推导产生式为止

根节点是文法的起始符号，产生式的左部作为子树的树根，产生式的右部作为子树的树叶，重复上述过程，直到没有非终结符可以生成子树了，最终就生成了一个句子。把树的叶子节点称之为这个推导所对应的句型（句子）。

⁎ 如此构造的语法树，其全体树叶（自左至右）恰好是给定的句型（句子）。

⁎ 句型、句子和语法树示例：

例 2.12 1. 证明 $(T/F+F)*i$ 是一个句型（表达式型）

2. 画出该句型的语法树。

算数表达式文法

$$E \to T \mid E+T \mid E-TT \to F \mid T*F \mid T/FF \to i \mid (E)$$

1. 证明 $(T/F+F)*i$ 是一个句型（表达式型）
2. 画出该句型的语法树。

证明：⁎ 句型 $(T/F+F)*i$ 的语法树构造：

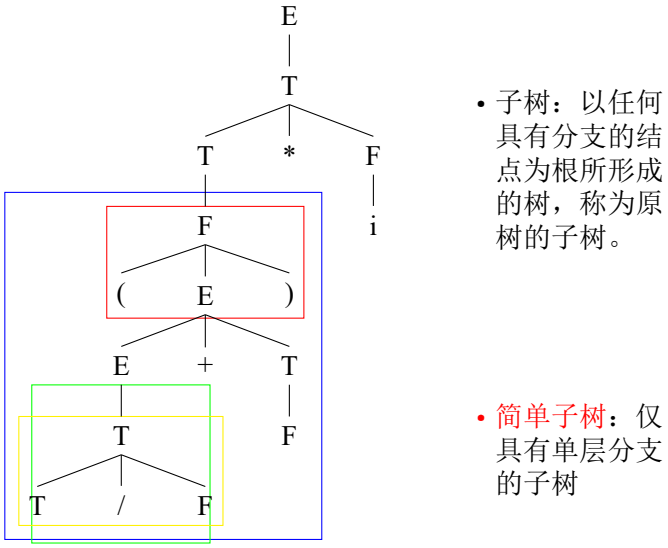


图 2.3: 语法树构造

$$E \Rightarrow T \Rightarrow T*F \Rightarrow F*F \Rightarrow (E)*F \Rightarrow (E+T)*F \Rightarrow (T+T)*F \Rightarrow (T/F+T)*F \Rightarrow (T/F+F)*F \Rightarrow (T/F+F)*i$$

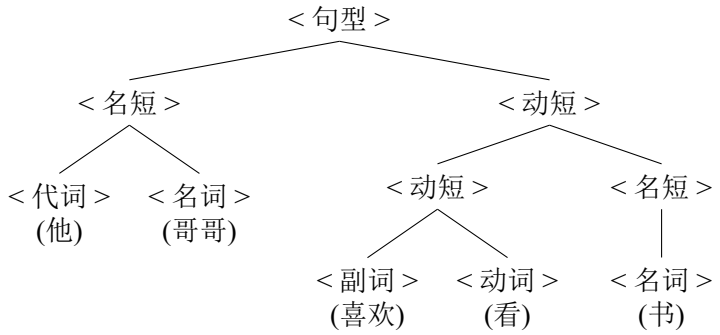
即：$E \overset{+}{\Rightarrow} (T/F + F) * i$

定义 2.28 子树：以任何具有分支的结点为根所形成的树，称为原树的子树。

定义 2.29 简单子树：仅具有单层分支的子树。

需要注意的是，2.3中红色圈出的部分不是子树，而是树片段，子树的叶子必须是原树的叶子。蓝色和绿色圈出的部分也不是简单子树，黄色圈出的子树才是简单子树。

例 2.13 图 2.4 为一个中文句型的语法树：



- 短语 – 他哥哥 < 名短 >，喜欢看 < 动短 >，书 < 名词 >，喜欢看书 < 动短 >，他哥哥喜欢看书 < 句子 >
- 简单短语 – 他哥哥，喜欢看，书
- 句柄 – 他哥哥 (最左边的简单短语!)

图 2.4: 中文句型的语法树

为什么要定义句柄？请读者思考这个问题。

2.3.3 短语、简单短语和句柄

* 设文法 G(Z),x y 是一个句型，则：

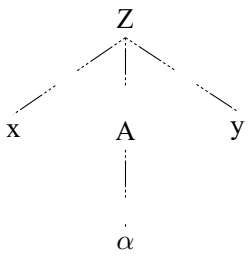


图 2.5: G(Z) 的语法树

定义 2.30 短语——若 $Z \overset{+}{\Rightarrow} xAy \overset{+}{\Rightarrow} x\alpha y$，则 α 是句型 $x\alpha y$ 关于 A 的短语 (A 是 α 的名字);

* 任一子树的树叶全体(具有共同祖先的叶节点符号串) 皆为短语；

定义 2.31 简单短语——若 $Z \overset{+}{\Rightarrow} xAy \Rightarrow x\alpha y$，则 α 是句型 $x\alpha y$ 关于 A 的短语 (A 是 α 的名字);

简单短语是通过一步推导出来的 * 任一子树的树叶全体(具有共同父亲的叶节点符号串) 皆为简单短语；

定义 2.32 句柄——一个句型的最左简单短语称为该句型的句柄

例 2.14 图 2.5 为一个算术表达式 (型) 的语法树:

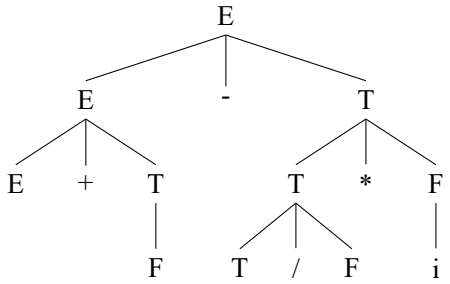


图 2.6: 算术表达式 (型) 的语法树

句型：$E+F-T/F*i$

短语：$E+F-T/F*i, E+F, F, T/F*i, T/F, i$

简单短语：$F, T/F, i$

句柄：F

⁎ 一类典型的综合例题：

例 2.15 $G(S): S \to aAcBe; A \to Ab|b; B \to d$

⁎ 给定符号串 α:aAbcde

1. 证明 $\alpha = $ aAbcde 是一个句型;
2. 画出句型 α 的语法树;
3. 指出 α 中的短语、简单短语和句柄

1. 证明 α 是由起始符号可以推导出的一个符号串就可以了
   (a) $S \Rightarrow$aAcBe $\Rightarrow$ aAbcBe $\Rightarrow$aAbcde
   (b) 语法树如右图:
   (c) 短语、简单短语和句柄:
   短语:aAbcde, Ab, d
   简单短语:Ab, d
   句柄:Ab
2. 每使用一条规则就把他所对应的子树片段画出来，一层一层画，就可以把这个树画出来

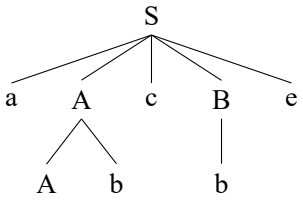


图 2.7: G(S) 的语法树

2.4 两种特性文法

设有文法:$G(Z) = (V_N, V_T, Z, P)$

2.4.1 递归文法

定义 2.33 设 $A \in V_N, x, y \in (V_N + V_T)^*$, 则; 若 $A \overset{+}{\Rightarrow} xAy$,: 称文法具有递归性;

A 是一个非终结符，若 A 经过若干次推导可以得到 xAy，x 和 y 是由终结符和非终结符构成的集合上的符号串，则称 A 所在的文法具有递归性。A 经过若干次推导，推导得到的中间结果里面仍然含有 A，就是 A 又生成了 A。如果在上下文无关文法中，A 可以无限的推导，这样我们得到了无限生成字符串的手段。

特别: 若 $A \to A\alpha$, 称文法具有直接左递归性;

$A \to \alpha A$, 称文法具有直接右递归性。

如果文法中存在这样的产生式，其左部和右部都含有非终结符 A，并且 A 在产生式右部的最左端，称这样的文法叫直接左递归文法。同理，如果产生式在左部和右部同时含有非终结符 A，并且 A 在产生式右部的最右端，称这样的文法叫直接右递归文法。

如：

直接左递归文法

G1(S) : S → Sb|a

直接右递归文法

G2(S) : S → bS|a

G1(S) 和 G2(S) 是正则文法

⁎ 递归文法是定义无限语言的工具（递归文法定义的语言有无限个句子）!!

2.4.2 二义性文法

定义 2.34 若文法中存在这样的句型，它具有两棵不同的语法树，则称该文法是二义性文法。

一个文法中含有一个句型，这个句型具有两棵及以上不同的语法树，即可以用不同的语法树来生成同一个句型，我们就称这个文法是二义性文法。

例 2.16 算术表达式的另一种文法:

算术表达式的另一种文法

G′(E) : E → E + E|E − E|E ∗ E|E/E| (E) | i;i(变量或常数)

这个算不算是一个算术表达式的文法？

∵ 句型 i + i ∗ i 有两棵不同的语法树 (右图):

左边的语法树先计算乘号，右边的语法树先计算加号。上述文法有没有考虑加减乘除的优先级？该文法没有考虑，这导致对于同一个表达式会有不同的分析。

如果用【例 2.10】给出的算术表达式文法，就不存在二义性。

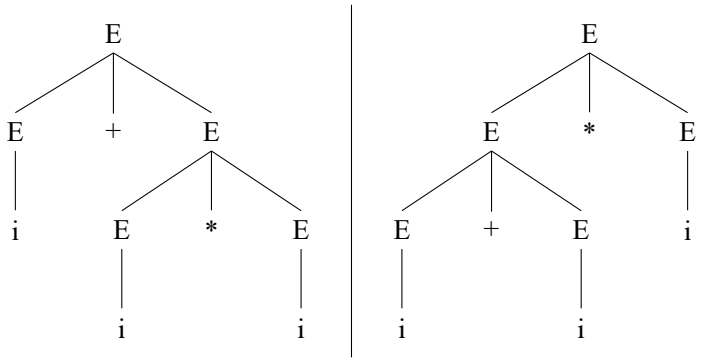


图 2.8: 不同的语法树

$\therefore G'(\mathbf{E})$ 是二义性文法。二义性文法会引起歧义，应尽量避免之！

2.5 文法的等价变换

2.5.1 文法的等价性

变换的依据是文法的等价性

定义 2.35 设 G1、G2 是两个文法，若 L(G1)=L(G2)，则称 G1 与 G2 等价，记作 G1≡G2。

如果文法 G1 和 G2 生成的语言是完全相同的，那么 G1 和 G2 这两个文法就是等价的。
即：文法的等价性是指他们所定义的语言是一样的

例 2.17 G1 和 G2 分别对应两个文法，G1 是一个直接右递归文法，G2 是一个直接左递归文法

文法规则

$G1: S->aS|a;$
$G2: S->Sa|a$

$L(G_1) = \{a,\ aa, aaa,\ \ldots\} = \{a^n \mid n \geqslant 1\}$
$L(G_2) = \{a, aa,\quad aaa, \ldots\} = \{a^n \mid n \geqslant 1\}$
$\therefore L(G_1) = L(G_2)$ 即 $G_1 \equiv G_2$
【注】一个语言，其描述文法并不唯一。
一个语言可以对应无数组文法描述，从这个角度说，如果一个文法描述了一个语言，那么我们是否有其他文法可以描述同样的语言呢？我们可以找到另外一个文法，这个文法没有歧义。

2.5.2 文法变换方法

在实际工作中，人们总是希望定义一种语言的文法尽可能地简单；另外，某些常用的语法分析技术也会对文法提出一定的要求或限制。为了适应上述要求，有时需要对文法进行必要的改写—通常称为文法变换，当然改写后的文法要与原文法等价。（文法变化具有技巧性）
这里重点介绍三类变换：

1. 删除无用的产生式（文法的化简）；
2. 删除 ε 产生式；（文法的右部是一个空串，我们要把他删除掉）

3. 常用的三种文法变换方法：（是我们一些常见的文法变换方法，这种方法比我们前面公式化要简单很多）
   ①必选项法；
   ②可选项法；
   ③重复可选项法。
   这三种方法会贯穿我们后面语法分析的部分。

2.5.3 文法变换方法 1

I 文法的化简
文法化简是指消除如下无用产生式：

1. 删除 A->A 形式的产生式 (自定己)；
2. 删除不能从其推导出终结符串的产生式 (不终结)；
3. 删除在推导中永不使用的产生式 (不可用)。

刚开始写文法的时候很难写出没有冗余的文法，当存在上述情况时如何处理？对于自定己产生式很容易找到，下面说一下后两种情况的处理方法。
⁎ 删除不终结产生式的算法：

1. 构造能推导出终结符串的非终结符集 V_{VT} :
   (a) 若有 $A->\alpha$ 且 $\alpha \in V_T^*$; 则令 $A \in V_{VT}$;
   (b) 若有 $B->\beta$ 且 $\beta \in (V_T + V_{VT})^*$; 则令 $B \in V_{VT}$;
   (c) 重复步骤① ②, 直到 V_{VT} 不再扩大为止。
2. 删除不在 V_{VT} 中的所有非终结符 (连同其产生式)。

想消除掉不终结的产生式，但是我们不容易求不终结的产生式，那我们可不可以求出哪些产生式是终结的。这个算法定义了 V_{VT}。

①对于 $A->\alpha$ 且 $\alpha \in V_T^*$,A 可以推出终结符，则把 A 放入 V_{VT}, 把产生式右部是终结符串的变量放入 V_{VT} 中

②B->β 这个产生式的右部是由终结符或 V_{VT} 中元素构成的符号串，我们就把 B 看作可终结的，把 B 放入 V_{VT} 中

③这个过程反复不断的执行，直到 V_{VT} 不再扩大为止, 如果这个符号不在 V_{VT} 中，我们就称 C 为不终结产生式的左部，我们把 C 和包含 C 的所有产生式删除。

⁎ 删除不可用产生式的算法：

1. 构造可用的非终结符集 V_{US}
   (a) 首先令 $Z \in V_{US}$;(Z 为文法开始符号)
   (b) 若有 $Z \overset{+}{\Rightarrow} \ldots A \ldots$, 则令 $A \in V_{Us}$;
   (c) 重复步骤②, 直到 V_{VT} 不再扩大为止。
2. 删除不在 V_{VT} 中的所有非终结符 (连同其产生式)。

不可用产生式就是由起始符号不能推导出来的非终结符所对应的产生式，我们称这样的产生式为不可用产生式。

找不可用产生式和不终结的产生式，他们的思路是一模一样的，一个是从终结符开始找，不可用是从起始符号开始找，只是方向有区别，算法是一样的。

首先定义一个集合 V_{VT}, 文法的开始符号都放在 V_{VT} 中，如果 Z 经过若干次推导，将推导出的符号串中的非终结符放入 V_{VT} 中，重复执行这样的过程直到 V_{VT} 不再扩展。最后，删除不在 VUS 中的所有非终结符连同其产生式。

运用上面三条规则来做一下例 2.16。

例 2.18 化简下述文法：

文法规则

G(S) : S → Be|Ec
A → Ae|e|A
B → Ce|Af
C → Cf; D → f; G → b

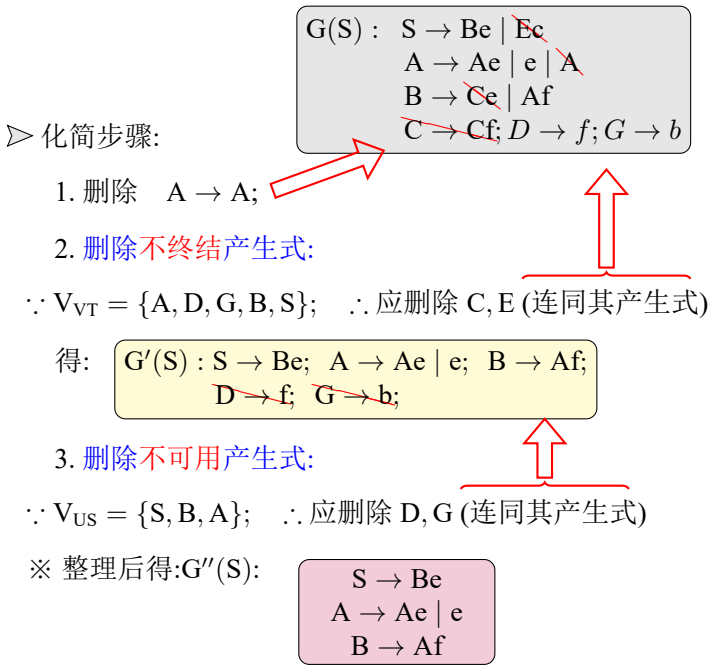


图 2.9: 文法化简示意图

2. 哪些非终结符对应的是不可终结的符号串？ CE
3. 起始符号到达不了的非终结符 DG

II 删除 ε 产生式

※ 假定文法 G(Z);$\varepsilon \equiv$ L(G) 【算法】

1. 首先构造可以推出空串的非终结符集:V_ε

①若有 A → ε; 则令 $A \in V_\varepsilon$

②若有 B → $A_1 \ldots A_n$ 且全部 $A_i \in V_\varepsilon$; 则令 B ∈ V_ε;

③重复步骤① ②, 直到 V_ε 不再扩大为止。

2. 删除 G(Z) 中的 A → ε 形成的产生式;

3. 依次改写 G(Z) 中的产生式 A → $X_1X_2 \ldots X_n$: 若有 $X_i \in V_\varepsilon$, 则用 $(X_i \mid \varepsilon)$ 替换之 (一个分裂为两个);

※ 若有 j 个 $X_i \in V_\varepsilon$, 则一个产生式将分裂为 2^j 个！

V_ε 中的非终结符，经过若干次推导可以推导出空串

①首先找直接能推出空串的产生式，把他的左部加入 V_ε

②假设产生式右部都是能推导出空串的符号，即 $A_1 \dots A_n$ 且全部 $A_i \in V_\varepsilon$，那么产生式左部所对应的非终结符放入 V_ε 中③重复上述过程，直到 V_ε 不再扩大为止。

消除文法中含有空符号串的产生式，找到文法中所有 $A \to \varepsilon$，再看 A 在哪些产生式中被使用了，如果这个产生式右部含有的符号恰巧是 V_ε 中的，就做如下的替换：把 X_i 写成 X_i 或 。

※ 若有 j 个 $X_i \in V_\varepsilon$, 则一个产生式将分裂为 2^j 个！

例 2.19 $G(S): S \to aAbc|bS \quad A \to dABe|\varepsilon; B \to A|b$

我们要求任何一个表达式的右部都不能含有空串

(1) 求解 $V_\varepsilon = \{A, B\}$

(2) 删除 ε 产生式 得:

含有 V_ε 元素的产生式

$S \to aAbc \mid bS;\ A \to dABe;\ B \to A \mid b$

(3) 改写含有 V_ε 中元素得产生式:

$\because S \to a(A \mid \varepsilon)bc$ $\quad \therefore S \to aAbc \mid abc$

$\because A \to d(A \mid \varepsilon)(B \mid \varepsilon)e$ $\quad \therefore A \to dABe \mid dBe \mid dAe \mid de$

$\because B \to (A \mid \varepsilon)$ $\quad \therefore B \to A$

※ 综合 G′:

$S \to aAbc|abc|bS$
$A \to dABe|dBe|dAe \mid de$
$B \to A \mid b$

图 2.10: 文法改写示意图

1. 我们要检查这些规则的右部，哪些元素是属于 V_ε 的
2. 先把第一条中的 A 替换为 (A|ε)，这种组合有两种再把第二条中的 A 和 B 替换为 (A|ε) 和 (B|ε)，这种组合有四种最后得到 G′(S) 和 G(S) 是等价的，并且 G′(S) 不包含产生 ε 的产生式

III 常用的三种文法变换方法：

文法的写作上有很多方便的形式来帮助我们把一些文法表达的更简便。

※ 基本思想：扩展文法，引进新的描述符号：

( ) 圆括号；[ ] 方括号；{ } 花括号

①必选项法（圆括号法）令 $(\alpha \mid \beta) = \alpha$ 或者 β

例如：有 $A \to a\alpha|a\beta$

可变换成：$A \to a(\alpha|\beta)$

也可：$A^A \to A' \quad ; \quad A' \to \alpha \mid \beta$

【注】此法又称提公因子法，利用此法可以使文法：

具有相同左部的各产生式首符号不同！

这两个文法规则是等价的，只不过我们把右部的 a 提出来了。

把圆括号的部分替换为 A′，就可以得到另外两个产生式，那为什么要这么做呢？读者可以思考一下。

②可选项法（方括号法）

令 $[\alpha] = \alpha$ 或者 ε 该方法表示方括号中的内容可选也可不选

例如：$S \to \alpha|\alpha\beta$

可变换成：$S \to \alpha[\beta]$

也可 $S \to \alpha S'; S' \to \beta \mid \varepsilon$

可以把方括号中的部分用 S′ 替换

S → if(B)S

S → if(B)S　else　S

B 为条件语句

可变换成:S → if(B)S[elseS]

或：S → if (B) SS′; S′ → else S | ε

把后面 else S 的部分用方括号圈起来，表达的是可选也可以不选

③重复可选项法（花括号法）

令 $\{\alpha\} = \varepsilon$ 或 α 或 $\alpha\alpha$ 或 $\alpha\alpha\alpha\ldots$

花括号代表无数种情况，表示由 α 构成的符号串包括空串

例如:S → A$\beta|\alpha$ 这个文法所生成的语言是一个什么样子的语言呢？

这个文法只有 A 使用 α 进行推导的时候才能生成终结符串，如果 A 推导出 Aβ 那这个推导就无法终止，

A → Aβ → Aβ^2 → Aβ^3 → $\cdots$ A$\beta^n = \alpha\beta^n$, n >= 0 可变换为：A → $\alpha\{\beta\}$

也可 A → αA′　;　A′ → βA′ | ε

⁎ 验证：

∵ 通过递推方法，可得:

A ⇒ Aβ ⇒ A$\beta\beta$ ⇒ A$\beta\beta\beta$ ⇒ … ⇒ $\alpha\beta*$

∴ 有 A → $\alpha\{\beta\}$; 或 A → αA′; A′ → βA′ | ε

【注】此方法常用来消除文法的直接左递归!

同样我们也可以用 A′ 来替换花括号中的内容，粉色和绿色的部分分别代表花括号的方式和文法改造的方式。白色部分的文法具有左递归，绿色部分的文法没有直接左递归，但是有递归的部分，是直接右递归。左递归和右递归在我们文法分析方法中导致的方法是完全不一样的。在后面文法分析一章会发现直接左递归文法很难分析。举个例子：比如用自上而下的分析，如果文法有直接左递归性，则采用最左推导时会导致死循环，所以在很多场景中我们要消除直接左递归。

2.6 形式语言的分类

Chomsky 把形式语言分为四类，分别由四类文法定义；四类文法的区别在于产生式的形式不同。

1. 0 型语言由 0 型文法定义
   - 产生式的形式为：$\alpha \to \beta$，又称无限制文法。
2. 1 型语言由 1 型文法定义
   - 产生式的形式为：xAy → xβy，又又称上下文有关文法。
3. 2 型语言由 2 型文法定义
   - 产生式的形式为：A → β，又称上下文无关文法。
4. 3 型语言由 3 型文法定义
   - 产生式的形式为：A → aB, A → a, A → ε，又称正规文法。

【注】四类语言为包含关系，且有 $L_0 \supset L_1 \supset L_2 \supset L_3$，编译处理中主要应用后两种文法。

在这门课中，用到的事 2 型语言和 3 型语言，上下文无关文法对应语法分析的部分，正规文法对应词法分析的部分。