第 5 章 上下文无关语言

从前面几章的讨论中了解到，正则文法所具有的描述能力是有限的。例如，在计算机高级程序设计语言的翻译中，正则文法只能对语言中组成单词的规则进行描述。通常称其为“词法”。也就是说，正则文法解决的是像标识符如何组成、整数如何组成、实数如何组成等问题。对于表达式、语句等更复杂的结构来说，正则文法就失去了描述能力。例如，在一般的算术表达式中，要求左括号与右括号是“配对”的，而正则文法不能表达这个“简单”的约束。事实上，不妨先忽略掉算术表达式中的其他部分，只留下“(”和“)”。这样，就可以用如下文法生成满足实际表达式要求的“良嵌套的”括号对序列。

G_{bra}：$S\rightarrow S(S)|\varepsilon$

考查该文法产生的如下形式的字符串，它们对应于一类简单嵌套的算术表达式：

$(^{n_1})^{n_1}(^{n_2})^{n_2}\cdots(^{n_h})^{n_h}$

用“0”表示“(”，用“1”表示“)”，则可得到下列已经在前面几章中所熟悉的字符串形式：

$0^{n_1}1^{n_1}0^{n_2}1^{n_2}\cdots 0^{n_h}1^{n_h}$

根据正则语言的泵引理容易证明，$L(G_{\text{bra}})$不是正则语言。经验表明，高级程序设计语言的绝大多数语法结构都可以用上下文无关文法(CFG)描述。因此，高级程序设计语言的规范说明及其编译是 CFG 的一个重要应用领域。用来描述高级程序设计语言的 **BNF**(Backus normal form，又叫 Backus-Naur form，**巴克斯范式**)就是 CFG 的一种特殊形式。CFG 的这种表达能力，以及计算机系统对于处理 CFG 的适应性，使得 CFG 和相应的上下文无关语言(CFL)在计算机的各种相关语言的处理、相关理论的研究中占有非常重要的地位。

5.1 上下文无关文法

按照 1.4 节所叙述的乔姆斯基体系定义的语言分类，如果存在一个 CFG G，使得 $L=L(G)$，则称语言 L 为上下文无关语言(CFL)。下面，先讨论文法和语言的派生。

根据定义 1-6，在 CFG 中，对于 $\forall \alpha\rightarrow\beta\in P$，均有 $\alpha\in V$ 成立。这就是说，对于 $\forall A\in V$，如果 $A\rightarrow\beta\in P$，则无论 A 出现在句型的任何位置，都可以将 A 替换成 β，而不考虑 A 的上下文。例如，设文法

G：$S\rightarrow AB$

$A\rightarrow aA|a$

$B \rightarrow bB \mid b$

虽然 $L(G)$中含有形如 $a^n b^n (n \geqslant 1)$的句子，但是，文法并没有要求 a 和 b 必须按照这种形式出现。实际上，对于任意的 $n \neq m$，也有 $a^n b^m \in L(G)$。也就是说，A 产生的 a 的个数并不受到 B 产生的 b 的个数的限制。所以

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

由于这种文法的语法变量在变换时不用考虑上下文，所以，它们对应的语言的分析（派生或者归约）相对来说就比较简单。目前，大多数高级程序设计语言的绝大多数语法特征都是上下文无关的，这使得这些语言的翻译系统比较容易实现。而自然语言的很多语言特征都不具有这种上下文无关的特性，而且除此之外，还有着与上下文有关的更复杂语义问题，所以处理起来要困难得多。

5.1.1　上下文无关文法的派生树

给定算术表达式的文法

G_{expl}：$E \rightarrow E+T \mid E-T \mid T$

$T \rightarrow T * F \mid T/F \mid F$

$F \rightarrow F \uparrow P \mid P$

$P \rightarrow (E) \mid N(L) \mid \mathrm{id}$

$N \rightarrow \sin \mid \cos \mid \exp \mid \mathrm{abs} \mid \log \mid \mathrm{int}$

$L \rightarrow L, E \mid E$

其中，id 表示基本的运算对象，它可以是一个表示变量的标识符，也可以是一个常数。“↑”表示幂运算，有时用“* *”表示。根据文法 G_{expl}，一个算术表达式可能有不同的派生和归约。例如，算术表达式$x+x/y \uparrow 2$就有多个不同的派生。图 5-1 给出了其中的 3 个不同派生。但是，这些不同的派生都表明，该表达式中的任何一个有意义的部分，都对应于语法变量的“同一个出现”。也就是说，这些派生只是对派生中产生的句型中的语法变量的替换顺序不同而已。如果把不同派生中用到的替换分别组成集合，这些集合是相等的。这表明，这些派生所“确认”的算术表达式 $x+x/y \uparrow 2$ 的结构是相同的。反过来，按照某一个给定的文法，一个句子的派生并不是唯一的。这就是说，一个句子可以对应多个派生。

(a)	(b)	(c)
$E \Rightarrow E+T$	$E \Rightarrow E+T$	$E \Rightarrow E+T$
$\Rightarrow T+T$	$\Rightarrow E+T/F$	$\Rightarrow T+T$
$\Rightarrow F+T$	$\Rightarrow E+T/F \uparrow P$	$\Rightarrow T+T/F$
$\Rightarrow P+T$	$\Rightarrow E+T/F \uparrow 2$	$\Rightarrow F+T/F$
$\Rightarrow x+T$	$\Rightarrow E+T/P \uparrow 2$	$\Rightarrow F+T/F \uparrow P$
$\Rightarrow x+T/F$	$\Rightarrow E+T/y \uparrow 2$	$\Rightarrow P+T/F \uparrow P$
$\Rightarrow x+F/F$	$\Rightarrow E+F/y \uparrow 2$	$\Rightarrow x+T/F \uparrow P$
$\Rightarrow x+P/F$	$\Rightarrow E+P/y \uparrow 2$	$\Rightarrow x+F/F \uparrow P$
$\Rightarrow x+x/F$	$\Rightarrow E+x/y \uparrow 2$	$\Rightarrow x+F/F \uparrow 2$
$\Rightarrow x+x/F \uparrow P$	$\Rightarrow T+x/y \uparrow 2$	$\Rightarrow x+F/P \uparrow 2$
$\Rightarrow x+x/P \uparrow P$	$\Rightarrow F+x/y \uparrow 2$	$\Rightarrow x+P/P \uparrow 2$
$\Rightarrow x+x/y \uparrow P$	$\Rightarrow P+x/y \uparrow 2$	$\Rightarrow x+P/y \uparrow 2$
$\Rightarrow x+x/y \uparrow 2$	$\Rightarrow x+x/y \uparrow 2$	$\Rightarrow x+x/y \uparrow 2$

图 5-1　算术表达式 $x+x/y \uparrow 2$ 的不同派生

显然，对这些派生的分析是比较烦琐的。为了能更清楚、直观地表达出句子的结构，这里略去派生中所有替换的顺序，而只保留所感兴趣的内容——替换。为此，可以用图 5-2 的树结构来表示这种对应关系。

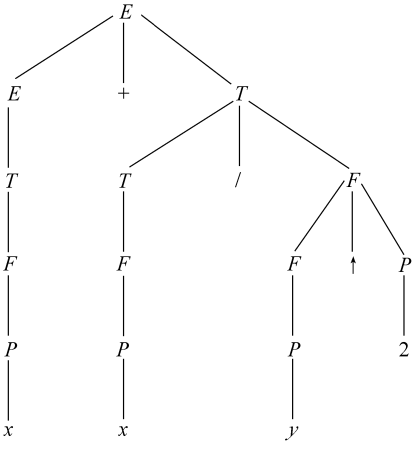


图 5-2 文法 G_{exp1} 的句子 $x+x/y\uparrow 2$ 的派生树

由于对一个输入串进行分析的目的就是要找出它的结构，从此角度讲，在表示句子的文法结构的问题上，派生不如树形式的表示更为清楚。这种树结构的形式称为文法 G_{exp1} 的一棵派生树。一般地，CFG 的派生树定义如下。

定义 5-1 设有 CFG $G=(V,T,P,S)$，G 的**派生树**(derivation tree)是满足如下条件的(有序)树(ordered tree)：

(1) 树的每个顶点有一个标记 X，且 $X\in V\cup T\cup\{\varepsilon\}$。

(2) 树根的标记为 S。

(3) 如果一个非叶子顶点 v 标记为 A，v 的儿子从左到右依次为 $v_1,v_2,\cdots,v_n$，并且它们分别标记为 $X_1,X_2,\cdots,X_n$，则 $A\rightarrow X_1X_2\cdots X_n\in P$。

(4) 如果 X 是一个非叶子顶点的标记，则 $X\in V$。

(5) 如果一个顶点 v 标记为 ε，则 v 是该树的叶子，并且 v 是其父顶点的唯一儿子。

派生树也称为**生成树**(derivation tree)、**分析树**(parse tree)、**语法树**(syntax tree)。按照这个定义，读者不难给出文法的一些派生树。请注意，这里定义的是 CFG 的派生树，显然，这种表达形式只适应于这种类型的文法。对于非 CFG，这种定义是无法适应的。所以，今后所讲的派生树，都是 CFG 的派生树。因此，可以省去“上下文无关文法的”这个限定词。

定义 5-2 设有文法 G 的一棵派生树 T，v_1，v_2 是 T 的两个不同的顶点，如果存在顶点 v，v 至少有两个儿子，使得 v_1 是 v 的较左儿子的后代，v_2 是 v 的较右儿子的后代，则称顶点 v_1 在顶点 v_2 的左边，顶点 v_2 在顶点 v_1 的右边。

定义 5-3 设有文法 G 的一棵派生树 T，T 的所有叶子顶点从左到右依次标记为 X_1，$X_2,\cdots,X_n$，则称符号串 $X_1X_2\cdots X_n$ 是 T 的**结果**(yield)。

根据这个定义，一个文法可以有多棵派生树，它们可以有不同的结果。所以，为了明确起见，对于任意一个 CFG G，可以称“G 的结果为 α 的派生树”为 G 的对应于句型 α 的派生树，简称为**句型 α 的派生树**。

定义 5-4　满足定义 5-1 中除了第(2)条以外各条的树叫**派生子树**(derivation subtree)。如果这个子树的根标记为 A,则称之为 A 子树。

定理 5-1　设 CFG $G=(V,T,P,S)$,$S\overset{*}{\Rightarrow}\alpha$ 的充分必要条件为 G 有一棵结果为 α 的派生树。

证明:为了证明方便,参考图 5-3,先证一个更为一般的结论:对于任意 $A\in V$,$A\overset{*}{\Rightarrow}\alpha$ 的充分必要条件为 G 有一棵结果为 α 的 A 子树。

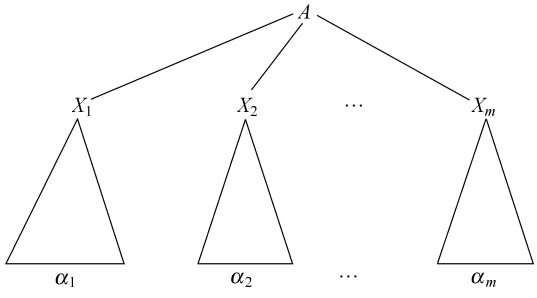


图 5-3　A 子树

充分性。设 G 有一棵结果为 α 的 A 子树,对这棵 A 子树的非叶子顶点的个数 n 施归纳,证明 $A\overset{*}{\Rightarrow}\alpha$ 成立。

当 $n=0$ 时,结论显然成立。当 $n=1$ 时,该 A 子树是一个二级子树。假设此树的叶子顶点的标记从左到右依次为 $X_1,X_2,\cdots,X_m$,由定义 5-1 的第(3)条,必有 $A\to X_1X_2\cdots X_m\in P$。注意到该子树的结果为 α,所以,$X_1X_2\cdots X_m=\alpha$,故 $A\overset{*}{\Rightarrow}\alpha$,即结论对 $n=1$ 成立。

设 $n\leqslant k(k\geqslant 1)$时结论成立,往证当 $n=k+1$ 时结论成立。设 A 子树有 $k+1$ 个非叶子顶点,根顶点 A 的儿子从左到右依次为 $v_1,v_2,\cdots,v_m$,并且它们分别标记为 $X_1,X_2,\cdots,X_m$,由定义 5-1 的第(3)条,必有 $A\to X_1X_2\cdots X_m\in P$。设分别以 $X_1,X_2,\cdots,X_m$ 为根的子树的结果依次为 $\alpha_1,\alpha_2,\cdots,\alpha_m$,其中,当 X_i 为一个叶子的标记时,取 $\alpha_i=X_i$。显然分别以 $X_1,X_2,\cdots,X_m$ 为根的子树的非叶子顶点的个数均不大于 k,由归纳假设:

$$X_1\overset{*}{\Rightarrow}\alpha_1$$

$$X_2\overset{*}{\Rightarrow}\alpha_2$$

$$\vdots$$

$$X_m\overset{*}{\Rightarrow}\alpha_m$$

且

$$\alpha=\alpha_1\alpha_2\cdots\alpha_m$$

从而

$$\begin{aligned}A&\Rightarrow X_1X_2\cdots X_m\\&\overset{*}{\Rightarrow}\alpha_1X_2\cdots X_m\\&\overset{*}{\Rightarrow}\alpha_1\alpha_2\cdots X_m\\&\vdots\\&\overset{*}{\Rightarrow}\alpha_1\alpha_2\cdots\alpha_m\end{aligned}$$

即结论对 $n=k+1$ 成立。由归纳法原理,结论对任意的 n 成立。

必要性。设 $A \overset{*}{\Rightarrow}\alpha$，现施归纳于派生步数 n，证明存在结果为 α 的 A 子树。

当 $n=0$ 时，结论显然成立。当 $n=1$ 时，由 $A\Rightarrow\alpha$ 知 $A\to\alpha\in P$。令 $\alpha=X_1X_2\cdots X_m$，则有图 5-4 所示的 A 子树。所以，结论对 $n=1$ 成立。

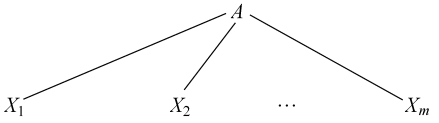


图 5-4 $A\Rightarrow\alpha$ 对应的 A 子树

设 $n\leqslant k(k\geqslant 1)$时结论成立，往证当 $n=k+1$ 时结论也成立。令 $A \overset{k+1}{\Rightarrow}\alpha$，则有

$$
\begin{aligned}
A &\Rightarrow X_1X_2\cdots X_m\\
&\overset{*}{\Rightarrow}\alpha_1X_2\cdots X_m\\
&\overset{*}{\Rightarrow}\alpha_1\alpha_2\cdots X_m\\
&\vdots\\
&\overset{*}{\Rightarrow}\alpha_1\alpha_2\cdots\alpha_m
\end{aligned}
$$

其中，对于任意的 i，$1\leqslant i\leqslant m$，$X_i\overset{*}{\Rightarrow}\alpha_i$。当 $X_i=\alpha_i$时，X_i是一个“只有一个顶点的 X_i子树”，X_i所标记的顶点既是叶子又是根；当 $X_i\overset{*}{\Rightarrow}\alpha_i$，所用的步数 $n_i\geqslant 1$ 时，必有 $n_i\leqslant k$，由归纳假设，存在以 α_i为结果的 X_i子树。即对于任意的 i，$1\leqslant i\leqslant m$，对应于 $X_i\overset{*}{\Rightarrow}\alpha_i$，存在以 α_i为结果的 X_i子树。由于 $A\Rightarrow X_1X_2\cdots X_m$，所以，$A\to X_1X_2\cdots X_m\in P$，从而我们可以得到图 5-3 所示的 A 子树的上半部分，然后再将所有的 X_i子树对应地接在 X_i所标识的顶点上，就可以得到图 5-3所示的树。显然，该树的结果为 α。所以，结论对 $n=k+1$ 成立。由归纳法原理，结论对任意的 n 成立。

综上所述，对于任意 $A\in V$，$A\overset{*}{\Rightarrow}\alpha$ 的充分必要条件为 G 有一棵结果为 α 的 A 子树。由于 $S\in V$，所以结论对 S 成立，从而定理得证。

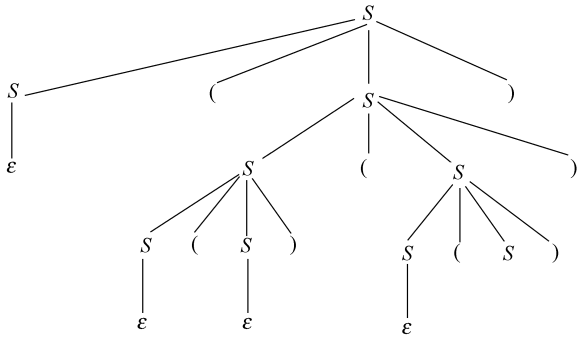
例 5-1 设 G_{bra}：$S\to S(S)\mid\varepsilon$，((()(())))和$(S)((S))$的派生树如图 5-5 所示。

由此例可以看出，派生树的结果可以是句子，也可以是句型。实际上，在定义 5-1 中，并没有要求派生树的叶子顶点的标记为文法的终结符号。

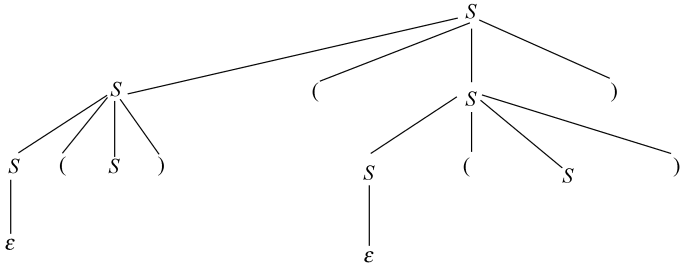
另外，定义 5-1 的第(5)条的意义在于避免一棵派生树中出现不必要的标记为 ε 的顶点。例如，如果没有这一条限制，对文法 G_{expl}产生的语言的句子 $x+x/y\uparrow 2$，可以构造出无穷多个“派生树”。图 5-6 就是其中之一。显然，这种类型的“派生树”中标记为 ε 的顶点不仅对树的结果没有贡献，而且使得该树变得非常复杂。这种复杂化只能给分析句型增加额外的负担。

定义 5-5 设有 CFG $G=(V,T,P,S)$，α 是 G 的一个句型。如果在 α 的派生过程中，每一步都是对当前句型的最左变量进行替换，则称该派生为**最左派生**(leftmost derivation)，每一步所得到的句型也可叫作**左句型**(left sentential form)，相应的归约叫作**最右归约**(rightmost reduction)；如果在 α 的派生过程中，每一步都是对当前句型的最右变量进行替换，则称该派生为**最右派生**(rightmost derivation)，每一步所得到的句型也可叫作**右句型**(right sentential form)，相应的归约叫作**最左归约**(leftmost reduction)。

在图 5-1 所给出的句子 $x+x/y\uparrow 2$ 的 3 种派生中，(a)为最左派生，(b)为最右派生，



(a)



(b)

图 5-5　派生树举例：(()(()))对应的派生树和(S)((S))对应的派生树

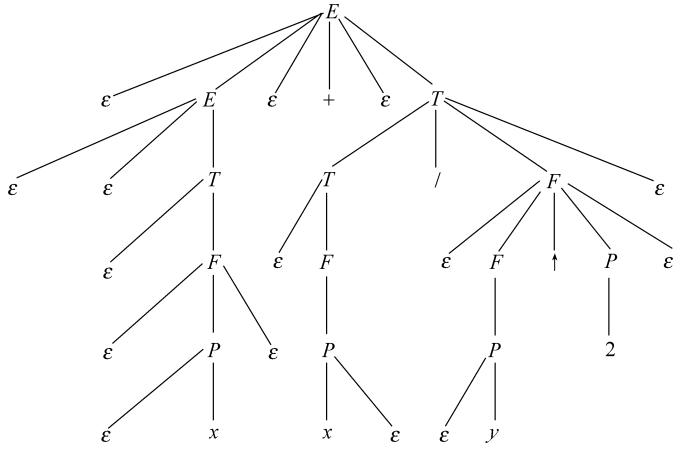


图 5-6　文法 G_{expl} 的句子 $x+x/y\uparrow 2$ 对应的“派生树”

(c)既不是最左派生也不是最右派生。

一般地，由于计算机系统在处理一个输入串时，通常都是从左至右进行的，这使得对句型的分析按照从左到右的顺序进行是比较自然的。所以，最右派生还称为**规范派生**(normal derivation)，规范派生产生的句型叫作**规范句型**(normal sentential form)，相应的归约叫作**规范归约**(normal reduction)。

定理 5-2　如果 α 是 CFG G 的一个句型，则 G 中存在 α 的最左派生和最右派生。

证明：现在对派生的步数 n 施归纳，证明对于任意 $A\in V$，如果 $A\overset{n}{\Rightarrow}\alpha$，则在 G 中存在对应的从 A 到 α 的最左派生：$A\overset{n左}{\Rightarrow}\alpha$。

当 $n=1$ 时，$A\Rightarrow\alpha$ 就是最左派生：$A\overset{左}{\Rightarrow}\alpha$。所以，结论成立。设 $n\leqslant k$ 时结论成立，

令 $A\overset{k+1}{\Rightarrow}\alpha$，则有

$$\begin{aligned}A &\Rightarrow X_1X_2\cdots X_m\\ &\overset{*}{\Rightarrow}\alpha_1X_2\cdots X_m\\ &\overset{*}{\Rightarrow}\alpha_1\alpha_2\cdots X_m\\ &\vdots\\ &\overset{*}{\Rightarrow}\alpha_1\alpha_2\cdots\alpha_m\end{aligned}$$

其中，$\alpha=\alpha_1\alpha_2\cdots\alpha_m$。对于任意的 i，$1\leqslant i\leqslant m$，$X_i\overset{*}{\Rightarrow}\alpha_i$。当 $X_i=\alpha_i$ 时，X_i 是由 A 开始的第一步派生得到的，所以，为了描述的统一起见，不妨认为，$X_i\overset{0}{\Rightarrow}\alpha_i$ 成立时，有 $X_i\overset{*左}{\Rightarrow}\alpha_i$ 成立；当 $X_i\neq\alpha_i$ 时，注意到 $X_i\overset{*}{\Rightarrow}\alpha_i$，所用的步数 $n_i\leqslant k$，由归纳假设，存在与之对应的 X_i 到 α_i 的最左派生：$X_i\overset{*左}{\Rightarrow}\alpha_i$。从而

$$\begin{aligned}A &\overset{左}{\Rightarrow}X_1X_2\cdots X_m\\ &\overset{*左}{\Rightarrow}\alpha_1X_2\cdots X_m\\ &\overset{*左}{\Rightarrow}\alpha_1\alpha_2\cdots X_m\\ &\vdots\\ &\overset{*左}{\Rightarrow}\alpha_1\alpha_2\cdots\alpha_m\end{aligned}$$

所以，结论对 $n=k+1$ 成立。由归纳法原理，结论对任意的 n 成立。

设 α 是 CFG $G=(V,T,P,S)$的一个句型。由句型的定义，$S\overset{n}{\Rightarrow}\alpha$。由于 S 是 V 中的一个元素，由上述证明，$S\overset{n左}{\Rightarrow}\alpha$。

同理可证，句型 α 有最右派生。定理得证。

用反证法并参考图 5-1 与图 5-2，读者不难证明下列结论。

定理 5-3 如果 α 是 CFG G 的一个句型，α 的派生树与最左派生和最右派生是一一对应的，但是，这棵派生树可以对应多个不同的派生。

5.1.2 二义性

定理 5-3 指出了派生树与最左派生和最右派生的一一对应关系，那么，句型和派生树又有什么样的关系呢？

考查文法

$$\begin{aligned}G_{exp2}：\ &E\to E+E\mid E-E\mid E/E\mid E*E\mid E\uparrow E\mid(E)\mid N(L)\mid \mathrm{id}\\ &N\to \sin\mid\cos\mid\exp\mid\mathrm{abs}\mid\log\mid\mathrm{int}\\ &L\to L,E\mid E\end{aligned}$$

$x+x/y\uparrow 2$ 是该文法的一个句子，图 5-7 给出了该句子关于文法 G_{exp2} 的 3 个不同的最左派生。显然，这 3 个不同的最左派生所表达出的句子 $x+x/y\uparrow 2$ 的“意思”是不相同的：在派生(a)中，$x+x/y\uparrow 2$ 中的第一个 x 是由句型 $E+E$ 中的第一个 E 派生出来的，$x/y\uparrow 2$ 则是

由该句型中的第二个 E 派生出来的，句子表达的意义是 x 加上 $x/y\uparrow 2$；进一步地，$x/y\uparrow 2$ 中的 x 是由句型 $x+E/E$ 中的第一个 E 产生的，而 $y\uparrow 2$ 则是由句型 $x+E/E$ 中的第二个 E 产生的，因此 $x/y\uparrow 2$ 表示 x 除以 y 的平方（$y\uparrow 2$）。在派生(b)中，句型 E/E 中的第一个 E 生成了 $x+x$，第二个 E 生成了 $y\uparrow 2$，所以，这个派生所表达的句子 $x+x/y\uparrow 2$ 的意思却是 $x+x$ 除以 $y\uparrow 2$。通过类似的分析可以发现，派生(c)所表达的句子 $x+x/y\uparrow 2$ 的意思是 $x+x$ 除以 y 所得的商的平方，也就是 $((x+x)/y)\uparrow 2$。它们所对应的派生树分别如图 5-8 中的(a)，(b)，(c)所示。

$E\Rightarrow E+E$	$E\Rightarrow E/E$	$E\Rightarrow E\uparrow E$
$\Rightarrow x+E$	$\Rightarrow E+E/E$	$\Rightarrow E/E\uparrow E$
$\Rightarrow x+E/E$	$\Rightarrow x+E/E$	$\Rightarrow E+E/E\uparrow E$
$\Rightarrow x+x/E$	$\Rightarrow x+x/E$	$\Rightarrow x+E/E\uparrow E$
$\Rightarrow x+x/E\uparrow E$	$\Rightarrow x+x/E\uparrow E$	$\Rightarrow x+x/E\uparrow E$
$\Rightarrow x+x/y\uparrow E$	$\Rightarrow x+x/y\uparrow E$	$\Rightarrow x+x/y\uparrow E$
$\Rightarrow x+x/y\uparrow 2$	$\Rightarrow x+x/y\uparrow 2$	$\Rightarrow x+x/y\uparrow 2$
(a)	(b)	(c)

图 5-7　句子 $x+x/y\uparrow 2$ 关于文法 G_{exp2} 的 3 个不同的最左派生

实际上，读者还可以找出文法 G_{exp2} 派生出的句子 $x+x/y\uparrow 2$ 对应的其他的不同派生树。按照这里的分析，不同的最左派生（派生树）表达出句子（句型）的不同含义。这就是说，按照所给的文法 G_{exp2}，句子 $x+x/y\uparrow 2$ 有多种不同的意义。显然，一般在利用文法定义语言，或者对语言进行分析时，是不希望这种情况发生的。句子的多义性会带来许多麻烦。可用如下定义描述这种现象。

定义 5-6　设有 CFG $G=(V,T,P,S)$，如果存在 $w\in L(G)$，w 至少有两棵不同的派生树，则称 G 是**二义性的**(ambiguity)；否则，G 为非二义性的。

上面所给的文法 G_{exp2} 是二义性的，而文法 G_{exp1} 派生出的句子 $x+x/y\uparrow 2$ 对应的派生树是唯一的。实际上，对于任意的 $w\in L(G_{exp1})$，G_{exp1} 派生出的句子 w 对应的派生树都是唯一的。也就是说，G_{exp1} 是无二义性的。然而 G_{exp1} 和 G_{exp2} 是等价的：

$$L(G_{exp1})=L(G_{exp2})$$

这表明，对于一个语言，在产生它的众多文法中，有的可能是二义性的，有的则可能是非二义性的。没有一个一般的方法来证明一个文法是不是二义性的。也就是说，判定任给 CFG G 是否为二义性的问题是一个**不可解的**(unsolvable)**问题**。

例 5-2　下列是描述高级程序设计语言的 if 语句的不同文法，其中，G_{ifa} 是二义性的，G_{ifm} 和 G_{ifh} 是为了消除 G_{ifa} 的二义性而对其进行改造的结果。实际上，这里的问题是 else 与哪一个 if 匹配的问题。在 G_{ifm} 和 G_{ifh} 中都规定了“就近匹配”的原则。

G_{ifa}：$S\rightarrow$ if E then S else S | if E then S

G_{ifm}：$S\rightarrow U|M$

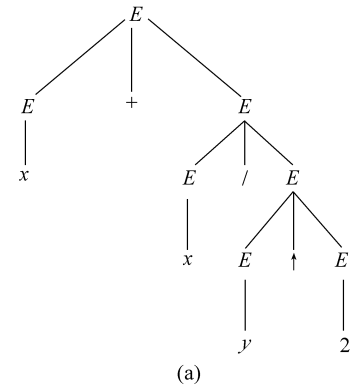
$U\rightarrow$ if E then S

$U\rightarrow$ if E then M else U

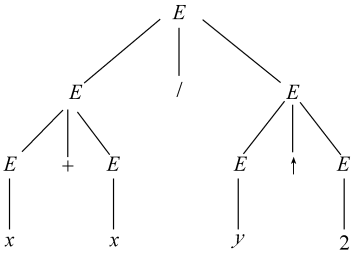
$M\rightarrow$ if E then M else $M|S$

G_{ifh}：$S\rightarrow TS|CS$

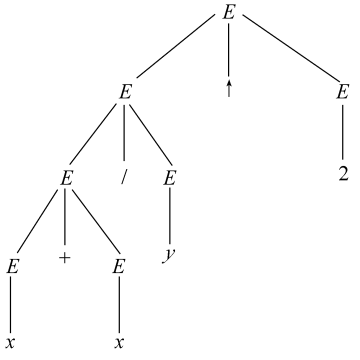
$C\rightarrow$ if E then



(a)



(b)



(c)

图 5-8 句子 $x+x/y\uparrow 2$ 对应的不同派生树

$T \rightarrow CS$ else

在文法 G_{ifm} 中，M 表示一个匹配的 if 语句(即含 else)，U 表示一个非匹配的 if 语句(即不含 else)，该文法除了迫使 then 和 else 之间必须有一个匹配的语句之外，还需要执行最长匹配原则才能完全地消除二义性。另外，也可以要求它的 M 产生式的第二个候选式 S 表示赋值语句、循环语句……等非 if 语句，以此来避免一个 U 被归约成 M。对文法 G_{ifh}，也要求在分析中增加最长匹配原则才能完全地消除二义性。这就是说，文法 G_{ifm} 和 G_{ifh} 都没有完全消除二义性(请读者考虑这两个文法的二义性问题)。要想彻底地消除二义性，还需要使用其他的一些限制，如这里提到的“最长匹配原则”和“M 的第二个候选式 S 是非 if 语句”等文法本身并未表达出来的限制。

解决二义性文法的方法还有一些，例如，给匹配确定优先级、增加标志等。

现在的问题是，是否所有的语言都有对应的非二义性文法呢？

例 5-3　设 $L_{\text{ambiguity}}=\{0^n1^n2^m3^m\mid n,m\geqslant 1\}\cup\{0^n1^m2^m3^n\mid n,m\geqslant 1\}$。

可以用如下文法产生语言 $L_{\text{ambiguity}}$，

G：$S\rightarrow AB\mid 0C3$

$A\rightarrow 01\mid 0A1$

$B\rightarrow 23\mid 2B3$

$C\rightarrow 0C3\mid 12\mid 1D2$

$D\rightarrow 12\mid 1D2$

不难找到句子 00112233 的如下两个不同的最左派生：

$S\Rightarrow AB$

$\Rightarrow 0A1B$

$\Rightarrow 0011B$

$\Rightarrow 00112B3$

$\Rightarrow 00112233$

$S\Rightarrow 0C3$

$\Rightarrow 00C33$

$\Rightarrow 001D233$

$\Rightarrow 00112233$

读者很容易画出这两个最左派生对应的不同派生树，因此，G 是二义性的。实际上，对于 $L_{\text{ambiguity}}$ 中形如 $0^n1^n2^n3^n(n\geqslant 1)$ 的句子，都有不同的派生树存在。可以证明，语言 $L_{\text{ambiguity}}$ 不存在非二义性的文法。

定义 5-7　如果语言 L 不存在非二义性文法，则称 L 是**固有二义性的**(inherent ambiguity)，又称 L 是**先天二义性的**。

定义 5-6 和定义 5-7 表明，文法可以是二义性的，语言可以是固有二义性的。一般地，不说文法是固有二义性的，也不说语言是二义性的。

5.1.3　自顶向下的分析和自底向上的分析

对于一个给定的文法，要想判定一个符号串是否为该文法的句子，需要考查是否可以从该文法的开始符号派生出此符号串。注意到归约是派生的逆过程，也可以考查这个符号串是否可以归约成文法的开始符号。第一种分析方法称为自顶向下的分析方法，第二种分析方法称为自底向上的分析方法。

例如，给定文法如下：

$S\rightarrow aAb\mid bBa$

$A\rightarrow aAb\mid bBa$

$B\rightarrow d$

判断 *aabdabb* 是否为该文法的句子。按照自顶向下的分析思路，需要寻找该句子的一个派生：

$S\Rightarrow aAb\Rightarrow aaAbb\Rightarrow aabBabb\Rightarrow aabdabb$

这个派生过程对应于相应的派生树从根到叶子的生长过程。如图 5-9 所示，从