形式语言理论

冯志伟 （昆明五中）

形式语言理论对于计算机程序语言和编译程序的研究有着重要意义。本文综述了N．Chomsky 等人在这方面所做的工作，通俗地介绍形式语言理论中四种类型的文法（有限状态文法、上下文无关文法、上下文有关文法和0型文法）以及与之相应的四种自动机（有限自动机、后进先出自动机、线性有界自动机和图灵机）。可供计算机软件和数理语言学工作者参考。

形式语言理论是计算机科学的一个重要领域。1956年，N. Chomsky 在研究自然语言的工作中，提出了文法的数学模型，为形式语言理论奠定了基础[2]。此后二十年来，在生产实践的推动之下，这门新学科得到了迅速发展，取得了不少成果。本文根据国外有关资料，对形式语言理论作一概述。

一、形式语言理论中语言和文法的概念

自然语言、计算机程序语言和其它人造语言都是负荷信息的符号体系，我们把它们统称为语言[30]。在形式语言理论中，语言被看成是一个抽象的数学系统，我们把它一般地定义为按一定规律构成的句子或符号串(string)的有限的或无限的集合，记为 L. N. Chomsky 和 G. A. Miller 指出："这样的定义把自然语言和逻辑与程序理论中的人造语言都包括进去了。"[11]。

每个句子或符号串的长度是有限的，它们由有限个符号相互毗连而构成。构成语言的有限个符号的集合，叫做字母表，记为V. 不包含任何符号的符号串，叫做空句子或空符号串，记为ε。

如果V是一个字母表，那么，我们把由

整型表达式。

5. PRIORITY(V)

这是一个标准过程。它将整型表达式V的值作为当前路径的优先级。

6. WAIT(S)

这是一个标准过程。它用来控制多条路径间的同步。其作用是，延迟当前路径的执行并把它送入延迟队列，直到其它路径执行SEND(S)以唤醒它为止，其中S的类型是系统使用的信号类型，该类型和队列有关。

7. SEND

这是一个标准过程。如果和S有关的队列中没有被延迟的路径，该标准过程便不起作用；否则，便唤醒其中优先数最高的（若有几个这样的路径，则先唤醒等待最久的）一个路径，被唤醒的路径立即投入运行。

8. AWAITED(S)

这是一个布尔型的标准函数。它检查和S有关的队列中是否有被延迟的路径，如有，则结果值为 TRUE；否则，结果值为FALSE。

9. P(S), V(S)

这是P, V 操作标准过程。它们也可用来控制多条路径间的同步与互斥。

10. PAUSE(V)

这是一个标准过程。其作用是，将当前路径延迟V秒后再执行，其中的V是一个整型表达式。

V 中的符号构成的全部句子（包括空句子 ε）的集合，记为 V^*；而把 V^* 中除 ε 之外的一切句子的集合，记为 V^+。例如，如果 $V=\{a, b\}$，则有

$$V^*=\{\varepsilon, a, b, aa, ab, ba, bb, aaa, \cdots\}$$

$$V^+=\{a, b, aa, \cdots\}$$

但是，某语言的字母表 V 中的符号相互毗连而构成的符号串，并不一定都是某语言中的句子。例如，“the man saw the ball”（“人看球”）在英语中是正确的，而由同样符号构成的“the saw the man ball”在英语中却是不正确的。我们把前者叫做成立句子，后者叫做不成立句子，而要区别一种语言中的成立句子和不成立句子，我们就有必要把这种语言刻画出来，从而说明，在这一种语言中，什么样的句子是成立的，什么样的句子是不成立的(注)。

我们可以采用三种办法来刻画语言。

第一种办法是把语言中的全部成立句子穷尽地枚举出来。如果语言只包含有限个句子，要穷尽地枚举是容易办到的，而如果语言中的句子数目是无限的，单用枚举的办法就行不通了。而且，在很多场合，对于语言中某一个长度有限的句子，还可以采用一定的办法将其长度加以扩展。例如，下面的句子在英语中都是成立的：

① This is the cat.（这是猫）

② This is the cat that caught the rat.（这是抓老鼠的猫）

③ This is the cat that caught the rat that ate the cheese.（这是抓吃乳酪的老鼠的猫）

我们可以在句子①上加上任意个 that-从句，每加一个这样的从句就构成了一个新的更长的句子。到底能够加多少个 that-从句，只与讲话人的记忆力和耐心有关，而与语言本身的结构无关。在这个意义上可以说，我们能够加上无限个 that-从句而使句子保持成立。因此用简单枚举的办法来刻画语言显然也是不行的。

第二种办法是制定有限个规则来孳生语言中无限个句子。例如，上面三个句子可以这样来描述：

设 X 为一个初始符号，S 为句子，R 为 that-从句，则有重写规则：

$$X \longrightarrow S,$$

$$S \longrightarrow S \frown R$$

这里，$\longrightarrow$ 是重写符号，$\frown$ 是毗连符号。利用这两条规则，可以孳生出无限个带 that-从句的句子。这有限个刻画语言的规则，就叫文法，记为 G. 文法是有限个规则的集合，这些规则递归地孳生出数目是潜在地无限的句子，并排除语言中的不成立句子。我们把由文法 G 所刻画的语言 L，记为 $L(G)$.

第三种办法是提出一个装置来检验输入符号串，从而识别该符号串是不是语言 L 中的成立句子，如果是成立句子，这个装置就接收它，如果是不成立句子，就不接收它，这样的装置叫做自动机，它是语言的识别程序（recognizer），记为 R.

由此可见，刻画某类语言的有效手段是文法和自动机，文法用于孳生语言，而自动机则用于识别语言［17］、［25］。本文着重介绍几种主要文法和自动机。

从形式上讲，我们可以把文法定义为四元组

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

其中，V_N 是非终极符号的集合，这些符号不能处于孳生终点；V_T 是终极符号的集合，这些符号能处于孳生终点；显然，V_N 与 V_T 的并构成了 V，V_N 与 V_T 不相交，因而有

$$V=V_N \cup V_T$$

$$V_N \cap V_T=\phi \text{（}\phi\text{ 表示空集合）}$$

V_N 中的符号用大写拉丁字母表示，V_T 中的符号用小写拉丁字母表示。符号串用希腊字母表示（有时也可以用拉丁字母表中排在较

（注） 关于成立句子和不成立句子的概念，可参看：冯志伟，数理语言学简介，《计算机应用与应用数学》，1975年第4期，第36页。

后面的如 w 之类的小写拉丁字母来表示符号串）。

S 是 V_N 中的初始符号；

P 是重写规则，其一般形式为：

$$\varphi \longrightarrow \psi,$$

这里，φ 是 V^+ 中的符号串，ψ 是 V^* 中的符号串，也就是说，$\varphi \neq \phi$.

如果我们用符号 # 来表示符号串的界限，那么，我们可以从初始符号串 $\# S \#$ 开始，应用重写规则构成语言 $L(G)$ 的成立句子。例如，利用重写规则 $\# S \# \longrightarrow \# \varphi_1 \#$，（从 $\# S \#$ 构成新的符号串 $\# \varphi_1 \#$）再利用重写规则 $\# \varphi_1 \# \longrightarrow \# \varphi_2 \#$，（从 $\# \varphi_1 \#$ 构成新的符号串 $\# \varphi_2 \#$）一直到我们得到不能再继续重写的符号串 $\# \varphi_n \#$ 为止。这样得到的终极符号串 $\# \varphi_n \#$ 显然就是语言 $L(G)$ 的成立句子。

重写符号⟶读为“可重写为”，它要满足如下条件：

① ⟶不是自反的；

② $A \in V_N$ 当且仅当存在 φ，ψ 和 ω，使得 $\varphi A \psi \longrightarrow \varphi \omega \psi$；

③ 不存在任何的 φ，ψ 和 ω，使得 $\varphi \longrightarrow \psi \# \omega$；

④ 存在元素对 $(\chi_1, \omega_1), \cdots, (\chi_n, \omega_n)$ 的有限集合，使得对于一切的 φ，ψ，当且仅当存在 φ_1，φ_2 及 $j \leqslant n$ 时，$\varphi = \varphi_1 \chi_j \varphi_2$ 和 $\psi = \varphi_1 \omega_j \varphi_2$，那么，$\varphi \longrightarrow \psi$.

由此可见，文法包含着有限个规则 $\chi_i \longrightarrow \omega_i$，这些规则充分地确定了该文法全部可能的孳生方式。这样，用这有限个规则，就可以孳生出语言中无限个句子。

例如，在英语中，有如下的文法

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

$V_N = \{$NP, VP, T, N, V$\}$

$V_T = \{$the, man, boy, ball, saw, hit, took, …$\}$

$S = S$

P：

S ⟶ NP⌒VP	(i)
NP ⟶ T⌒N	(ii)
VP ⟶ V⌒NP	(iii)
T ⟶ the	(iv)
N ⟶ man, boy, ball, …等等	(v)
V ⟶ saw, hit, took, …等等	(vi)

这里，初始符号 S 表示句子，NP 表示名词短语，VP 表示动词短语，T 表示定冠词，N 表示名词，V 表示动词（注意不要跟表示字母表的那个 V 相混）。

利用这些重写规则，从初始符号 S 开始，我们可以孳生出英语的成立句子：“the man saw the ball”，“the man took the ball”，“the man hit the ball”，“the boy hit the ball”…等。

如“the man saw the ball”的孳生过程如下，后注重写规则的号码：

S	
NP⌒VP	(i)
T⌒N⌒VP	(ii)
T⌒N⌒V⌒NP	(iii)
the N⌒V⌒NP	(iv)
the man V⌒NP	(v)
the man saw NP	(vi)
the man saw T⌒N	(ii)
the man saw the N	(iv)
the man saw the ball	(v)

这种孳生过程，叫做推导史（derivational history）。其它语句亦同。

当然，这里的文法只是英语文法的一小片段，孳生出的语言，也只是英语的一小部分。

又如，提出文法：

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

$V_N = \{S\}$

$V_T = \{a, b, c\}$

$S = S$

P：

$S \longrightarrow aca$	(i)
$S \longrightarrow bcb$	(ii)

$$S \longrightarrow aSa \qquad \text{(iii)}$$

$$S \longrightarrow bSb \qquad \text{(iv)}$$

此文法可以孳生出所谓有中心元素的镜象结构语言，该语言句子分三部分：前面是若干个 a 及若干个 b 相毗连，中间是单个的符号 c，后面是在 c 后与前面成镜象关系的若干个 a 及若干个 b 的毗连，即 $abcba$，$bbacabb$，$ababacababa$，…。用 α 表示集合 $\{a, b\}$ 上的任意非空符号串，用 α^* 表示 α 的镜象，则这种语言可表示为 $\{\alpha c\alpha^*\}$。

如果要孳生符号串 $abbaacaabba$，那么，从 S 开始的推导史如下：

$$\begin{array}{cr} S & \\ a\ S\ a & \text{(iii)} \\ a\ b\ S\ b\ a & \text{(iv)} \\ a\ b\ b\ S\ b\ b\ a & \text{(iv)} \\ a\ b\ b\ a\ S\ a\ b\ b\ a & \text{(iii)} \\ a\ b\ b\ a\ a\ c\ a\ a\ b\ b\ a & \text{(i)} \end{array}$$

显然，由此文法孳生出的语言的符号串其数目是无限的。

下面研究定义由文法 G 所孳生的语言 $L(G)$。为此先引入表示 V^* 上的符号串之间关系的符号 $\underset{G}{\Rightarrow}$ 及 $\overset{*}{\underset{G}{\Rightarrow}}$.

如果 $\alpha \longrightarrow \beta$ 是 P 的重写规则，φ_1 及 φ_2 是 V^* 上的任意符号串，有

$$\varphi_1\alpha\varphi_2 \underset{G}{\Rightarrow} \varphi_1\beta\varphi_2,$$

（读为“在文法 G 中，$\varphi_1\alpha\varphi_2$ 直接推导出 $\varphi_1\beta\varphi_2$”）就说，应用重写规则 $\alpha \longrightarrow \beta$ 于符号串 $\varphi_1\alpha\varphi_2$，得到了符号串 $\varphi_1\beta\varphi_2$. 也就是说 $\underset{G}{\Rightarrow}$ 表示这两个符号串之间的直接推导关系。

假定 α_1. α_2, …, α_m 是 V^* 上的符号串，并且 $\alpha_1 \underset{G}{\Rightarrow} \alpha_2$, $\alpha_2 \underset{G}{\Rightarrow} \alpha_3$, …, $\alpha_{m-1} \underset{G}{\Rightarrow} \alpha_m$，那么，就说

$$\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m,$$

（读为“在文法 G 中，α_1 推导出 α_m”）。简言之，如果我们应用 P 中的若干个重写规则，由 α 得到 β，那么，我们就说，对于两个符号串 α 与 β，有 $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$.

于是文法 G 孳生的语言 $L(G)$ 可定义如下：

$$L(G) = \{w \mid w \text{ 在 } V_T^* \text{ 中，并且 } S \overset{*}{\underset{G}{\Rightarrow}} w\},$$

由此可见，一个符号串处在 $L(G)$ 中，要满足两个条件：

① 该符号串只包括终极符号；

② 该符号串能从 S 推导出来。

同一语言可以由不同的文法来孳生。如果 $L(G_1) = L(G_2)$，则文法 G_1 等价于文法 G_2.

二、文法的 Chomsky 分类

上面所定义的文法 $G = (V_N, V_T, P, S)$，其重写规则为 $\varphi \longrightarrow \psi$，并且要求 $\varphi \neq \phi$. 现在，我们加上一些限制：

限制 1：如果 $\varphi \longrightarrow \psi$，那么，存在 A，φ_1，φ_2，ω，使得 $\varphi = \varphi_1 A \varphi_2$，$\psi = \varphi_1 \omega \varphi_2$.

限制 2：如果 $\varphi \longrightarrow \psi$，那么，存在 A，φ_1，φ_2，ω，使得 $\varphi = \varphi_1 A \varphi_2$，$\psi = \varphi_1 \omega \varphi_2$，并且 $A \longrightarrow \omega$.

限制 3：如果 $\varphi \longrightarrow \psi$，那么，存在 A，φ_1，φ_2，ω，a，Q，使得 $\varphi = \varphi_1 A \varphi_2$，$\psi = \varphi_1 \omega \varphi_2$，$A \longrightarrow \omega$，并且 $\omega = aQ$ 或 $\omega = a$. 因而，$A \longrightarrow aQ$ 或 $A \longrightarrow a$.

限制 1 要求文法的重写规则都具有形式 $\varphi_1 A \varphi_2 \longrightarrow \varphi_1 \omega \varphi_2$，这种重写规则在上下文 $\varphi_1 — \varphi_2$ 中给出 $A \longrightarrow \omega$. 如果符号串的长度定义为符号串中的符号数并用 $|\psi|$ 和 $|\varphi|$ 分别表示符号串 ψ 和 φ 的长度，则显然有 $|\psi| \geqslant |\varphi|$。由于在重写规则 $\varphi_1 A \varphi_2 \longrightarrow \varphi_1 \omega \varphi_2$ 中，每当 A 出现于上下文 $\varphi_1 — \varphi_2$ 中时，可用 ω 替换 A，因此，此种文法叫做上下文有关文法或 1 型文法。

限制 2 要求文法的重写规则都具有形式 $A \longrightarrow \omega$，这时上下文 $\varphi_1 — \varphi_2$ 是空的，在运用重写规则时不依赖于单个的非终极符号 A

所出现的上下文，因此，叫做上下文无关文法或2型文法。

限制3要求文法的重写规则具有形式 A $\longrightarrow a$Q 或 A$\longrightarrow a$，其中 A 和 Q 是非终极符号，a 是终极符号，这种文法叫做有限状态文法或3型文法有时也可叫做正则文法。

没有上述限制的文法，叫做0型文法。

0型文法孳生的语言叫做0型语言，上下文有关文法、上下文无关文法或有限状态文法孳生的语言，分别叫做上下文有关语言、上下文无关语言或有限状态语言，也可以分别叫做1型语言、2型语言或3型语言。

由于从限制1到限制3的限制条件是逐渐增加的，因此，不论对于文法或对于语言来说，都有

0型$\supseteq$1型$\supseteq$2型$\supseteq$3型。

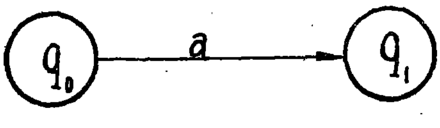
上面所介绍的文法的四种类型及其所孳生的语言，是首先由 N. Chomsky 提出的[5]、[12]，因此，人们把它们称之为 Chomsky 分类。

下面，进一步说明这四种类型的文法。

1. 有限状态文法

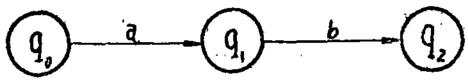
有限状态文法的重写规则为 A $\longrightarrow a$Q 或 A$\longrightarrow a$（A$\longrightarrow a$ 只不过是 A$\longrightarrow a$Q 当 Q$=\phi$ 时的一种特例），如果把 A 和 Q 看成不同的状态，那么，由重写规则可知，当从状态 A 到状态 Q 时，可孳生出一个终极符号 a. 这样，可把有限状态文法想象为一种孳生装置，每次能孳生一终极符号，而每一终极符号与一个特定的状态相联系。我们改用字母 q 来表示状态，这样，如果这种孳生装置原先处于某一状态 q_i，孳生出一个终极符号后，就可以到状态 q_j，在状态 q_j 再孳生出一个终极符号后，就可以到状态 q_k…等等。这种情况，可用状态图来表示[4]。

例如，如果这种孳生装置原先处于某状态 q_0，孳生出一个终极符号 a 后，转入状态 q_1，那么，其状态图为：



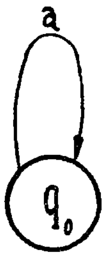
它孳生出的语言是 a。

如果这种孳生装置原先处于状态 q_0，孳生出终极符号 a 后，转入状态 q_1，在状态 q_1 再孳生出终极符号 b 后，转入状态 q_2，那么，其状态图为：



它孳生出的语言是 ab.

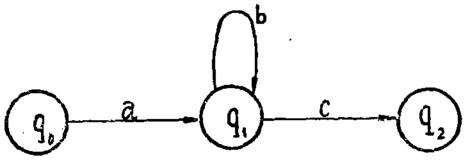
如果这种孳生装置处于状态 q_0，孳生出终极符号 a 后又回到 q_0，那么，其状态图为：



这种状态图叫做圈（loop）。

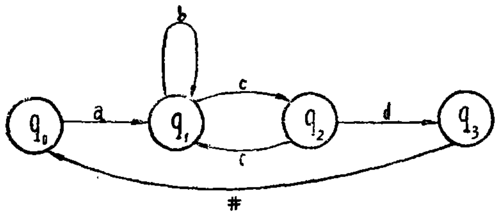
它孳生出的语言是 a，aa，aaa，$aaaa$，…等等。

如果这种孳生装置处于状态 q_0，孳生出终极符号 a 后转入状态 q_1，在状态 q_1，或者孳生出终极符号 b 后再回到 q_1，或者孳生出终极符号 c 后转入状态 q_2，那么，其状态图为：



它孳生出的语言是 ac，$a$bc，$a$bbc，…等等。

这种孳生装置在孳生了若干个终极符号之后，还可能转回到前面的状态，构成一个大封闭圈。例如下面的状态图：



它可以孳生如 $abbcccd\#$，$abcd\#$，$abcccccd\#$，…等终极符号串。这里 $\#$ 表示符号串的终点，q_0 既是初始状态，又是最后状态。

可见，给出一个状态图，就可以按着图中的路，始终顺着箭头所指的方向来孳生句子。当达到图中的某一状态时，我们可以沿着从这一状态引出的任何一条路前进，不管这条路在前面的孳生过程中是否已经走过；在从一个状态到另一个状态时，可以容许若干种走法；状态图中还可以容许有任意有限长度的任意有限数目的圈。这样的孳生装置，在数学上叫有限状态 Марков 过程。

状态图是有限状态文法的形象表示法，因此，我们可以根据状态图写出其相应的有限状态文法。

例如，与最后一个状态图相应的有限状态文法如下：

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

$V_N=\{q_0, q_1, q_2, q_3\}$

$V_T=\{a, b, c, d, \#\}$

$S=q_0$

P:

$q_0 \longrightarrow aq_1$

$q_1 \longrightarrow bq_1$

$q_1 \longrightarrow cq_2$

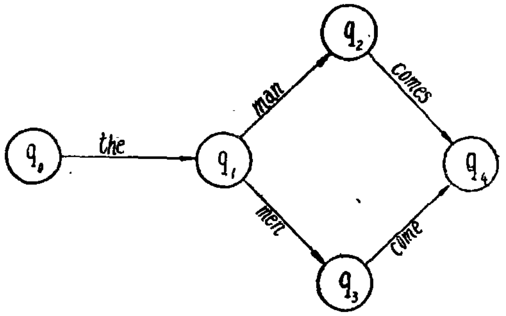
$q_2 \longrightarrow cq_1$

$q_2 \longrightarrow dq_3$

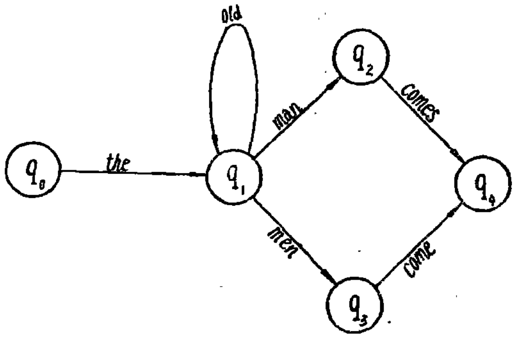
$q_3 \longrightarrow \#q_0$

在这种文法中，q_0，q_1，q_2，q_3 表示状态，它们都是非终极符号。不难看出，其重写规则符合于有限状态文法重写规则的形式。

用有限状态文法来刻画自然语言是不合适的[3]、[29]。例如，可以提出一个有限状态文法来孳生两个英语句子："the man comes"（"人来了"）及"the men come"（"人们来了"），其状态图如下：

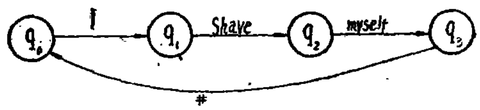


如果我们在状态 q_1 处加一个圈，则其状态图变为：

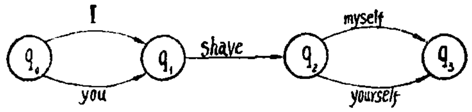


这时，这个有限状态文法除了能孳生上面两个句子之外，还可孳生"the old man comes"，"the old old man comes"，…，以及"the old men come"，"the old old men come"，…等句子。可以看出，其中有一些句子在英语中是不成立的。

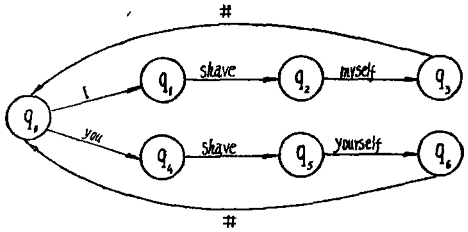
又如，我们可提出一个有限状态文法来孳生英语句子"I shave myself"（"我自己刮胡子"），其状态图如下：



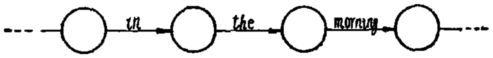
如果除"I shave myself"之外，还要孳生句子"you shave yourself"（"你自己刮胡子"），则可提出这样的状态图：



但是，这个状态图也可孳生“I shave yourself”这样的不成立句子。为了防止孳生出这样的不成立句子，我们必须把状态图分成两路，使它们无法联系。但这样一来，状态图就变得复杂了。



如果要把“in the morning”这样的短语加在前面的句子上，那么，又要在上面那个状态图中加上如下的状态图：



这样一来，状态图就变得更加复杂了。

如果要孳生出英语中的一篇文章、一本书籍，那么，其状态图就不知要多复杂!

由此可见，有限状态文法作为一种刻画自然语言的模型是不行的。

N. Chomsky 指出，有一些由非常简单的符号串构成的语言，不能由有限状态文法孳生[3]。例如：

(i)、ab, aabb, aaabbb, ….可以把这种语言表示为 $L_1=\{a^nb^n\}$，其中，$n\geqslant 1$.

(ii)、aa, bb, abba, baab, aaaa, bbbb, aabbaa, abbbba, …. 如果用 α 表示集合 {a, b} 上的任意非空符号串，用 α^* 表示 α 的镜象，那么，这种语言可表示为 $L_2=\{\alpha\,\alpha^*\}$.

(iii)、aa, bb, abab, aaaa, bbbb, aabaab, abbabb, …. 如果用 α 表示集合 {a, b} 上的任意非空符号串，那么，这种语言可表示为 $L_3=\{\alpha\,\alpha\}$.

L_1，L_2 和 L_3 都不能由有限状态文法孳生出来，可见，这种文法的孳生能力不强。

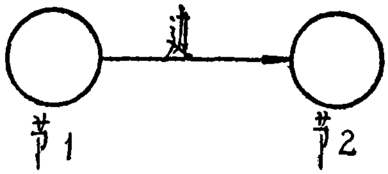
2 上下文无关文法

上下文无关文法 P 中的重写规则的形式是 $A\longrightarrow\omega$，其中，A 是单个的非终极符号，ω 是任何异于 ε 的符号串，即 $|A|=1\leqslant|\omega|$.

例如，前面提到过的孳生带中心元素的镜象结构语言的文法，它的重写规则的左边都是单个的非终极符号 S，右边都是异于 ε 的符号串，因而它是上下文无关文法。

上下文无关文法的推导过程是由推导树 (derivation tree) 来描述的。

树是图论中的一个概念。树由边(edge)和节（node）组成，它是由边连接着的节的有限集合。如果一个边由节 1 指向节 2，那么我们就说，边离开节 1 而进入节 2。如图所示：



树要满足如下三个条件：

① 树中要有一个没有任何边进入的节，称之为根（root).

② 对于树中的每一个节，都要有一系列的边与根连接着。

③ 除根以外，树中的每一个节，都只能有一个边进入它，因此，树中没有圈。

如果有一个边离开给定的节 m 而进入节 n，那么，所有的节 n 的集合叫做节 m 的直接后裔 (direct descendant)。如果有一系列的节 n_1，n_2，…，n_k，使得 $n_1=m$，$n_k=n$，并且对于每一个 i，n_{i+1} 是 n_i 的直接后裔，那么，节 n 就叫做节 m 的后裔(descendant)。规定，一个节是它自身的后裔。

对于树中的每一个节，可以把其直接后裔按顺序排列起来。设 n_1 和 n_2 是节 n 的直接

后裔，而n_1在n_2之前，就说，n_1及它的各个后裔处于n_2及它的各个后裔的左侧。每一个节都是根的后裔。如果n_1和n_2是节，而且其中一个节又不是另一个节的后裔，那么，它们二者必定是某个节的后裔，这样，n_1和n_2中的一个就处于另外一个的左侧。

设$G=(V_N, V_T, P, S)$是上下文无关文法，如果有某个树满足下列条件，它就是G的推导树：

① 每个节有一个标号，这个标号是V中的符号；

② 根的标号是S；

③ 如果节n至少有一个异于其本身的后裔，并有标号A，那么，A必定是V_N中的符号；

④ 如果节n_1，n_2，…，n_k是节n的直接后裔，从左排列起来，其标号分别为A_1，A_2，…，A_k，那么，

$$A \longrightarrow A_1A_2\cdots A_k$$

必是P中的重写规则。

例如，我们来考虑文法

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

$V_N=\{A, S\}$

$V_T=\{a, b\}$

$S=S$

P：

$S \longrightarrow aAS$

$A \longrightarrow SbA$

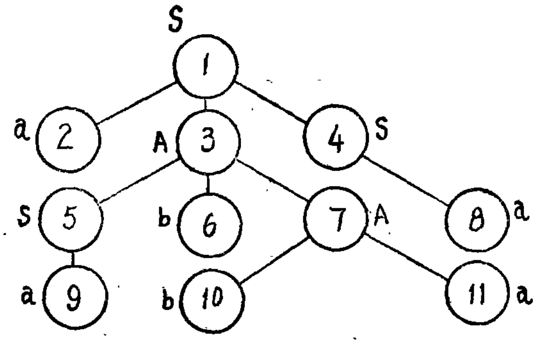
$A \longrightarrow SS$

$S \longrightarrow a$

$A \longrightarrow ba$

这个文法的五个重写规则，左边都是单个的非终极符号S或者A，右边都是异于ε的符号串，因而是上下文无关文法。

现在我们画出这个文法的推导树。为便于说明，我们用圆圈表示节，把节编上号码，把标号注在节的旁边，边的方向假定都是直接向下的，不用箭头标出，其推导树如下：

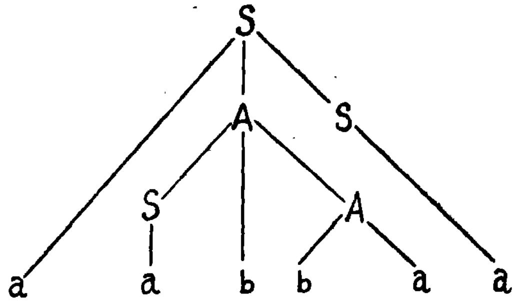


从这个推导树中可看出，节1是树的根，它的标号是S．节2，节3，节4是节1的直接后裔，节2处于节3和节4的左侧，节3处于节4的左侧。节10是节3的后裔，但不是直接后裔。节5处于节10的左侧。因为节3处于节4的左侧，而节11是节3的后裔，所以，节11处于节4的左侧。

1，3，4，5，7等节都有直接后裔。节1的标号为S，其直接后裔的标号从左算起为a、A和S，因而$S \longrightarrow aAS$是重写规则。节3的标号为A，其直接后裔的标号从左算起为S，b和A，因而$A \longrightarrow SbA$是重写规则。节4和节5的标号为S，它们每一个的直接后裔标号为a，因而$S \longrightarrow a$是重写规则。节7的标号为A，其直接后裔的标号从左算起为b和a，因而$A \longrightarrow ba$也是重写规则。由此可见，刚才画出的文法G的推导树满足推导树的要求条件。

在任何树中，总有一些节是没有后裔的，这样的节叫做叶（*leaf*）。如果从左到右读推导树中各个叶的标号，我们就可得到一个终极符号串，这个终极符号串叫做推导树的结果（*result*），可以证明，如果α是上下文无关文法$G=(V_N, V_T, P, S)$的结果，则$S \overset{*}{\underset{G}{\Rightarrow}} \alpha$．例如，在上述推导树中，各个叶从左到右的标号为2，9，6，10，11和8，它们的标号分别是a，a，b，b，a，a，则推导树的结果$\alpha=aabbaa$，因此，$S \overset{*}{\underset{G}{\Rightarrow}} aabbaa$.

将节及其编号去掉，上述推导树可简化为：



也可以把这个过程写为：

$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS$

$\Rightarrow aabbaS \Rightarrow aabbaa.$

如果改换推导过程，还可以得到该文法孳生的其它终极符号串。例如：

$S \Rightarrow aAS \Rightarrow abaS \Rightarrow abaa.$

$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS$

$\Rightarrow aabSSS \Rightarrow aabaSS \Rightarrow aabaaS$

$\Rightarrow aabaaa.$

上下文无关文法的孳生能力比有限状态文法强得多。N. chomsky 所指出的语言$L_1=\{a^nb^n\}$和语言$L_2=\{\alpha\alpha^*\}$，不能由有限状态文法孳生，但可以用上下文无关文法孳生，现在来孳生L_1和L_2这两种语言。

提出上下文无关文法

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

$V_N=\{S\}$

$V_T=\{a, b\}$

$S=S$

P：

$S\longrightarrow aSb$

$S\longrightarrow ab$

从S开始，运用第一个重写规则$(n-1)$次，然后再运用第二个重写规则1次，我们得到：

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow a^3Sb^3 \Rightarrow \cdots$

$\Rightarrow a^{n-1}Sb^{n-1} \Rightarrow a^nb^n.$

又提出上下文无关文法

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

$V_N=\{S\}$

$V_T=\{a, b\}$

$S=S$

P：

$S\longrightarrow aa$

$S\longrightarrow bb$

$S\longrightarrow aSa$

$S\longrightarrow bSb$

这样的文法可孳生语言$L_2=\{\alpha\alpha^*\}$。例如，如果要孳生语言$\{\alpha\alpha^*\}$中的符号串 abbbba，我们可这样来推导：

$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abbbba.$

可是，用上下文无关文法不能孳生语言$L_3=\{\alpha\alpha\}$.

再来考虑一种语言 $L=\{a^ncb^{2n}\}, n\geqslant 1$，这种语言可提出这样的上下文无关文法$G$来孳生：

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

$V_N=\{S, C\}$

$V_T=\{a, b, c\}$

$S=S$

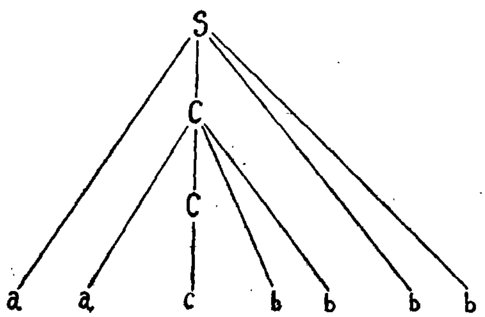
P：

$S\longrightarrow aCbb$

$C\longrightarrow aCbb$

$C\longrightarrow c$

例如，要孳生符号串$aacbbbb$，其推导树为



上下文无关文法的重写规则 $A\longrightarrow\omega$ 可以改写成范式。范式有两种，一种是 *chomsky* 范式，一种是 *Greibach*范式。

chomsky 范式是由 N. chomsky 提出

的[12]。他证明了，任何上下文无关语言，可由重写规则的形式为$A\longrightarrow BC$或$A\longrightarrow a$的文法孳生出来，这里，A，B和C是非终极符号，a是终极符号，这样的重写规则就叫做Chomsky范式。

利用Chomsky范式可把任何上下文无关文法的推导树简化为二元形式。例如，孳生语言$\{a^ncb^{2n}\}$的上下文无关文法可用下述方式变换为chomsky范式：

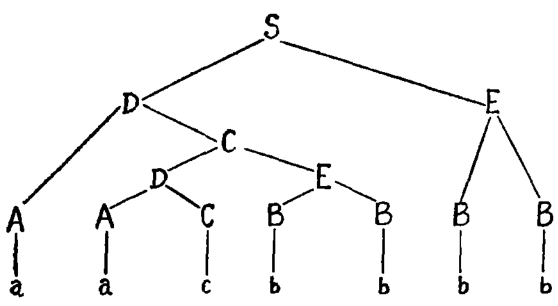
这个文法的重写规则为：

$$S\longrightarrow aCbb$$
$$C\longrightarrow aCbb$$
$$C\longrightarrow c$$

其中$C\longrightarrow c$是符合Chomsky范式要求，不必再变换。用$S\longrightarrow ACBB$及$A\longrightarrow a$，$B\longrightarrow b$替换$S\longrightarrow aCbb$，用$C\longrightarrow ACBB$及$A\longrightarrow a$，$B\longrightarrow b$替换$C\longrightarrow aCbb$。然后，再把$S\longrightarrow ACBB$，$C\longrightarrow ACBB$的右边换成二元形式，用$S\longrightarrow DE$，$D\longrightarrow AC$及$E\longrightarrow BB$替换$S\longrightarrow ACBB$，用$C\longrightarrow DE$，$D\longrightarrow AC$及$E\longrightarrow BB$替换$C\longrightarrow ACBB$。这样，便得到了如下符合Chomsky范式要求的文法的重写规则：

$S\longrightarrow DE$	$A\longrightarrow a$
$D\longrightarrow AC$	$B\longrightarrow b$
$E\longrightarrow BB$	$C\longrightarrow c$
$C\longrightarrow DE$	

用Chomsky范式，可将符号串$aacbbbb$的推导树简化成如下的二元形式：



Greibach范式是由S. A. Greibach提出的[18]。他证明了，任何上下文无关语言，可由重写规则的形式为$A\longrightarrow a\alpha$的文法孳生出来，这里，A是非终极符号，a是终极符号，α是非终极符号组成的符号串（可以为空的），这样的重写规则就叫做Greibach范式。例如，我们可以采用下面的办法把孳生语言$\{a^ncb^{2n}\}$的上下文无关文法的重写规则

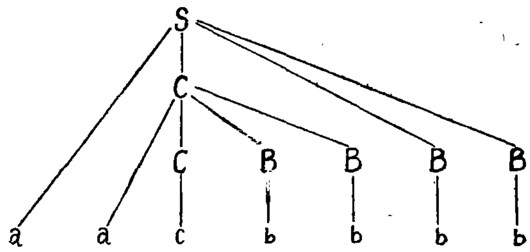
$$S\longrightarrow aCbb$$
$$C\longrightarrow aCbb$$
$$C\longrightarrow c$$

变换为Greibach范式。

$C\longrightarrow c$中的c后的符号串为空，符合Greibach范式的要求，不必再变换，只要用$S\longrightarrow aCBB$及$B\longrightarrow b$来替换$S\longrightarrow aCbb$，用$C\longrightarrow aCBB$及$B\longrightarrow b$来替换$C\longrightarrow aCbb$。便得到了如下符合Greibach范式要求的文法的重写规则：

$S\longrightarrow aCBB$	$C\longrightarrow c$
$C\longrightarrow aCBB$	$B\longrightarrow b$

用Greibach范式可将符号串$aacbbbb$的推导树写成如下形式：



前面讲过，每一个有限状态文法都是上下文无关的，但反之不一定。在上下文无关文法中，如果存在某一非终极符号A，具有性质$A\overset{*}{\underset{G}{\Longrightarrow}}\varphi A\psi$，这里，$\varphi$及$\psi$是非空符号串，这样，在其推导过程中，$A$自身就会嵌入到符号串$\varphi A\psi$中去，那么，就说，这个上下文无关文法是自嵌入的。N. Chomsky证明了，如果G是非自嵌入的上下文无关文法，那么，L (G) 就是有限状态语言。他又证明了，如果$L(G)$是上下文无关语言，那么，当且仅当文法G是具有自嵌入性质的

上下文无关文法时，$L(G)$才不是有限状态语言[5]、[12]、[20]。我们前面讨论过的$\{a^nb^n\}$、$\{aa^*\}$、$\{aca^*\}$以及$\{a^ncb^{2n}\}$等上下文无关语言，在它们的文法的重写规则中，以及在用文法来孳生符号串的过程中，都会出现$A \xRightarrow[G]{*} \varphi A \psi$这样的推导式，具有自嵌入性质，因此，这些语言都不可能是有限状态语言，而是自嵌入的上下文无关语言。可见，不是有限状态语言的上下文无关语言确实是存在的。

下面介绍最左推导的概念。如果在推导过程中作每一步推导时，符号串中被替换的非终极符号的左侧不再有非终极符号，那么，就说这种推导是最左推导。这就是说，如果在文法$G=(V_N, V_T, P, S)$中，$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$是最左推导，那么，对于$1 \leqslant i < n$，当由$\alpha_i \Rightarrow \alpha_{i+1}$时，我们能够把$\alpha_i$写为$x_iA_i\beta_i$，把$\alpha_{i+1}$写为$x_i\gamma_i\beta_i$，其中，$x_i$是$V^*_T$中的符号串，$\beta_i$是$V^*$中的符号串，$A_i$是$V_N$中的符号，$A_i \longrightarrow \gamma_i$是$P$中的生成式，$\alpha_{i+1}$是从$\alpha_i$中用$\gamma_i$替换$A_i$而推导出来的。

对于某个上下文无关文法$G=(V_N, V_T, P, S)$，如果$L(G)$中的同一个句子有两个或两个以上相异的最左推导，那么，我们就说，这个上下文无关文法是二义性的。

例如文法$G=(V_N, V_T, P, S)$

$V_N=\{S, A, B\}$

$V_T=\{a, b\}$

$S=S$

P:

$S \longrightarrow bA$	$S \longrightarrow aB$
$A \longrightarrow a$	$B \longrightarrow b$
$A \longrightarrow aS$	$B \longrightarrow bS$
$A \longrightarrow bAA$	$B \longrightarrow aBB$

这些重写规则都是Greibach范式，易于用它们来进行最左推导。$L(G)$的句子$aabbab$有如下两个相异的最左推导：

① $S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabB \Rightarrow aabbS \Rightarrow aabbaB \Rightarrow aabbab.$

② $S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabSB \Rightarrow aabbAB \Rightarrow aabbaB \Rightarrow aabbab.$

因此，这个文法G是二义性的。

N. Chomsky和M. P. Schützenberger证明了，一个任意的上下文无关文法是否有二义性的问题是不可判定的[13]。这是上下文无关文法的一个相当重要的性质，它告诉我们，对于一个上下文无关文法，我们找不到一般的过程来判定，在系统地检查了这个文法的全部规则之后，这个文法是不是有二义性。

3. 上下文有关文法

上下文有关文法P中的重写规则形式为$\varphi \longrightarrow \psi$，$\varphi$和$\psi$是符号串，并且要求$|\psi| \geqslant |\varphi|$，也就是$\psi$的长度不小于$\varphi$的长度。

语言$L=\{a^nb^nc^n\}$，它是由n个a，n个b和n个c相毗连着的符号串（$n \geqslant 1$）。提出文法G来孳生它。

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

$V_N=\{S, B, C\}$

$V_T=\{a, b, c\}$

$S=S$

P:

$S \longrightarrow aSBC$ (i)

$S \longrightarrow aBC$ (ii)

$CB \longrightarrow BC$ (iii)

$aB \longrightarrow ab$ (iv)

$bB \longrightarrow bb$ (v)

$bC \longrightarrow bc$ (vi)

$cC \longrightarrow cc$ (vii)

从S开始，用规则(i)$n-1$次，得到

$$S \overset{*}{\Rightarrow} a^{n-1}S(BC)^{n-1},$$

然后用规则(ii)1次，得到

$$S \overset{*}{\Rightarrow} a^n(BC)^n,$$

规则(iii)使我们可以把$(BC)^n$变换为(B^nC^n)，例如，如果$n=3$，有

$aaaBCBCBC \Rightarrow aaaBBCCBC \Rightarrow$
$aaaBBCBCC \Rightarrow aaaBBBCCC$.

这样，

$$S \overset{*}{\Rightarrow} a^n B^n C^n,$$

接着，用规则(iv) 1 次，得到

$$S \overset{*}{\Rightarrow} a^n b B^{n-1} C^n,$$

然后，用规则(v) $n-1$ 次，到得

$$S \overset{*}{\Rightarrow} a^n b^n C^n,$$

最后，用规则(vi) 1 次及规则(vii) $n-1$ 次，得到

$$S \overset{*}{\Rightarrow} a^n b^n c^n.$$

在这个文法中，它的 7 个重写规则的右边的符号数大于或等于左边的符号数，满足条件 $|\psi| \geqslant |\varphi|$，因此，这个文法是上下文有关文法。注意，上下文有关文法重写规则左边的φ是非空符号串，而有限状态文法及上下文无关文法的重写规则左边的A必须是V_N中单个的非终极符号，这是它们的不同之处。

N. Chomsky证明了，对于上下文有关文法G，如果X和B是G的符号串，在G中加上规则$XB \longrightarrow BX$而构成文法G'，那么，必然存在着一个上下文有关文法G^*等价于G'[5]。

据此，N. Chomsky进一步证明了，不是上下文无关语言的上下文有关语言是存在的[5]。例如语言$L=\{a^n b^m a^n b^m ccc\}$ $(m, n \geqslant 1)$就是这样的语言，如下的上下文有关文法可以孳生它：

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

$V_N=\{S, S_1, S_2, A, \bar{A}, B, \bar{B}, C, D, E, F\}$

$V_T=\{a, b, c\}$

$S=S$

P分四部分：

(Ⅰ) a) $S \longrightarrow C\ D\ S_1\ S_2\ F$

b) $S_2 \longrightarrow S_2\ S_2$

c) $\left\{\begin{array}{l} S_2F \longrightarrow BF \\ S_2B \longrightarrow BB \end{array}\right\}$

d) $S_1 \longrightarrow S_1S_1$

e) $\left\{\begin{array}{l} S_1\ B \longrightarrow AB \\ S_1\ A \longrightarrow AA \end{array}\right\}$

(Ⅱ) a) $\left\{\begin{array}{l} CDA \longrightarrow CE\bar{A}A \\ CDB \longrightarrow CE\bar{B}B \end{array}\right\}$

b) $\left\{\begin{array}{l} CE\bar{A} \longrightarrow \bar{A}CE \\ CE\bar{B} \longrightarrow \bar{B}CE \end{array}\right\}$

c) $E\alpha\beta \longrightarrow \beta E\alpha$

d) $E\alpha \longrightarrow D\alpha$

e) $\alpha D \longrightarrow D\alpha$

(Ⅲ) $CDF\alpha \longrightarrow \alpha CDF$

(Ⅳ)

a) $\left\{\begin{array}{l} A \longrightarrow a \\ \bar{A} \longrightarrow a \\ B \longrightarrow b \\ \bar{B} \longrightarrow b \end{array}\right\}$

b) $\left\{\begin{array}{l} CDF \longrightarrow CDc \\ CDc \longrightarrow Ccc \\ Cc \longrightarrow cc \end{array}\right\}$

这里，α与β取集合$\{A, B, F\}$中的值。

现在，从S开始来孳生语言$\{a^n b^m a^n b^m ccc\}$。

① 这样运用规则(Ⅰ)：用a) 1 次；对于$m \geqslant 1$，用b) $m-1$ 次；用c) m 次；对于$n \geqslant 1$，用d) $n-1$ 次；用e) n 次。可得到符号串：

$CD\alpha_1 \cdots \alpha_{n+m}F$。

这里，对于$i \leqslant n$，$\alpha_i=A$；对于$i>n$，$\alpha_i=B$。

② 这样运用规则(Ⅱ)：用a) 1 次，用b) 1次，如果$\alpha_i=A$，则$\bar{\alpha}_i=\bar{A}$，如果$\alpha_i=B$，则$\bar{\alpha}_i=\bar{B}$，这时得到：

$\bar{\alpha}_i CE\alpha_1 \cdots \alpha_{n+m}F$；

用c) $n+m$次，用d) 1 次，得到：

$\bar{\alpha}_1 C\alpha_2 \cdots \alpha_{n+m}FD\alpha_1$；

用e) $n+m$次，得到：

$\bar{\alpha}_1 CD\alpha_2 \cdots \alpha_{n+m}F\alpha_1$。

③ 再按②的办法，用规则（Ⅱ）$n+m-1$次，并得到：

$\bar{a}_1\cdots\bar{a}_{n+m}CDFa_1\cdots a_{n+m}$。

④ 用规则（Ⅲ）$n+m$次，得到：

$\bar{a}_1\cdots\bar{a}_{n+m}a_1\cdots a_{n+m}CDF$。

⑤ 这样运用规则（Ⅳ）：用a）$2(n+m)$次，用 b）3 次，这时便得到所要孳生的语言：

$a^nb^ma^nb^mccc$。

如果按另外的顺序来使用上述规则，就不会得到终极符号串 $a^nb^ma^nb^mccc$。这个终极符号串的形式是完全由第①步运用规则（Ⅰ）时所选择的n和m来决定的。规则（Ⅱ）和（Ⅲ）的作用是把形式为$CDXF$的符号串（X表示由符号A和B组成的符号串）改变为形式为$XXCDF$的符号串。规则（Ⅳ）的作用是把形式为$XXCDF$的符号串改变为终极符号串 $a^nb^ma^nb^mccc$。显而易见，这种语言是不能用上下文无关文法来孳生的。

N. Chomsky指出，语言$\{a^nb^nc^n\}$也是一种不能用上下文无关文法孳生的上下文有关语言[12]。

前面说过，N. Chomsky指出的不能用有限状态文法来孳生的语言 $L_3=\{\alpha\alpha\}$，也不能用上下文无关文法来孳生，但是，它可以用上下文有关文法来孳生。

现在来孳生语言 $L_3=\{\alpha\alpha\}$。为此，提出上下文有关文法

$G=(V_N,\ V_T,\ P,\ S)$

$V_N=\{S\}$

$V_T=\{a,\ b\}$

$S=S$

P：

$S\longrightarrow aS$ (i)

$S\longrightarrow bS$ (ii)

$\alpha S\longrightarrow \alpha\alpha$ (iii)

在规则（iii）中，α 是集合 $\{a,\ b\}$ 上的任意非空符号串。

例如，可以这样来孳生语言 $abbabb$：从S开始，用规则（i）1 次，得到 $S\overset{*}{\Rightarrow}aS$，用规则（ii）两次，得到$S\overset{*}{\Rightarrow}abbS$，最后用规则（iii）1 次，得到$S\Rightarrow abbabb$。

可见，上下文有关文法的孳生能力比有限状态文法和上下文无关文法都强。在描写自然语言方面，它可以处理有限状态文法和上下文法无关文法不能处理的一些问题[11]。但是，近年来由于转换文法（transformational grammar）的发展，在自然语言的描写中已经不大使用上下文有关文法了[11]、[28]。

4. 0型文法

0型文法的重写规则是$\varphi\longrightarrow\psi$，除了要求$\varphi\neq\phi$之外，它不受什么限制。N. Chomsky证明了，每一个0型语言都是符号串的递归可枚举集；并且证明，任何一个上下文有关语言同时又是0型语言，而且还存在着不是上下文有关语言的0型语言，因此，上下文有关语言应包含在0型语言之中，它是0型语言的子集合[5]。

0型文法是具有图灵机孳生能力的一种装置，对于0型文法和0型语言的研究，是递归函数论的一个组成部分[5]。关于图灵机，后面还要谈到。

三、接收语言的自动机

自动机是语言的识别程序。由文法G孳生出的语言的识别程序R，记为$R(G)$。$R(G)$能处理语言L中的符号串σ，并决定σ是不是文法G孳生出来的成立句子，如果$\sigma\in L(G)$，那么，$R(G)$都停止，这时就说，符号串σ被自动机接收了。因此，自动机又可看成语言的接收机[21]、[23]、[24]。这里，介绍与前面讨论过的那四种文法相对应的四种自动机：有限自动机、后进先出自动机、线性有界自动机及图灵机。

1. 有限自动机

有限自动机是有限状态语言的接收机，

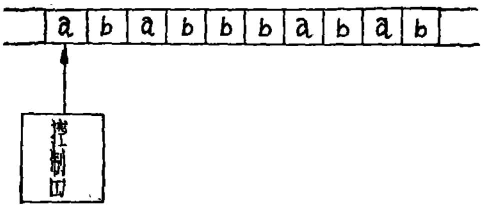
是四种自动机中最简单的一种[9]。

一个有限自动机M可以定义为五元组：

$M=(K,\ \Sigma,\ \delta,\ q_0,\ F)$，

其中，K是有限状态的非空集合，Σ是有限的输入字母表，δ是$K\times\Sigma$到K的映射，这种映射可同文法中的规则P相比拟，q_0是K中的初始状态，$F\subseteq K$是最后状态的集合。

有限自动机由控制器、带子及读数头组成，其结构原理可图示如下：



带子分为若干单元，每一单元记录Σ中的一个符号，这样，便可把输入符号串记录在带子上。读数头从最左边的符号开始扫描，每扫描一个输入符号就向右移动一个单元。控制器可以有K中的有限个状态，当M从位于最左边符号处的读数头开始工作时，控制器处于初始状态q_0，每扫描一个符号向右移动一个单元，控制器也就改变到一个新的状态。当输入符号串读完时，如果M处于最后状态，那么，就可以说，我们所输入的符号串被有限自动机接收了。

M的读数、移动、改变状态等工作，可用δ映射来表示。δ映射的一般形式为

$$\delta(q_i,\ a_j)=q_k。$$

它的意思是，对于K中的q_i，q_k以及Σ中的a_j，M在状态q_i时，扫描符号a_j，然后向右移动一个单元，同时状态改变到q_k。

δ映射中的a_j，也可以是带子中记录的输入符号串，我们用x表示。这时，δ映射的形式为

$$\delta(q_i,\ x)=q_k。$$

如果对于F中的某一个p，$\delta(q_0,\ x)=p$，那么，就说，符号串x被M接收了。所有被M接收的x的集合，记为$T(M)$。也就是说，

$$T(M)=\{x\mid\delta(q_0,\ x)在F中\}。$$

有限自动机是有限状态语言的接收机。N. Chomsky证明了，如果$G=(V_N,\ V_T,\ P,\ S)$是有限状态文法，那么，存在着有限自动机$M=(K,V_T,\delta,S,F)$，使得$T(M)=L(G)$。他又证明了，如果给出一个有限自动机M，那么，存在着有限状态文法G，使得$L(G)=T(M)$。[4]

有限自动机可以用状态图来表示。如果有限自动机处于状态q，扫描输入符号a，然后转入状态p，那么，我们用圆圈表示状态，从状态q转入状态p用带标号a的箭头表示，最后状态用双圈表示，初始状态用写着“开始”的箭头标出来。

例如，有这样的有限自动机M：

$M=(K,\ \Sigma,\ \delta,\ q_0,\ F)$

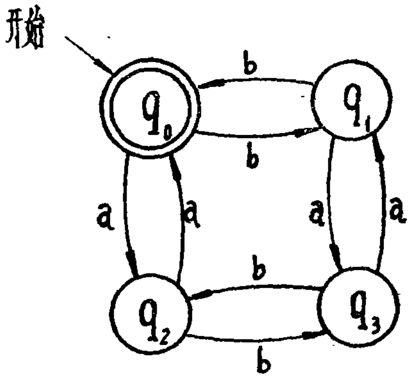
$K=\{q_0,\ q_1,\ q_2,\ q_3\}$

$\Sigma=\{a,\ b\}$

$F=\{q_0\}$

$\delta(q_0,\ a)=q_2$	$\delta(q_0,\ b)=q_1$
$\delta(q_1,\ a)=q_3$	$\delta(q_1,\ b)=q_0$
$\delta(q_2,\ a)=q_0$	$\delta(q_2,\ b)=q_3$
$\delta(q_3,\ a)=q_1$	$\delta(q_3,\ b)=q_2$。

其状态图如下：



这样的有限自动机可接收符号串是由偶数个a及偶数个b组成的语言。如果把这样的符号串$bbabab$输入到M中，由于$\delta(q_0,b)=q_1$及$\delta(q_1,\ b)=q_0$，所以，$\delta(q_0,\ bb)=q_0$，这时，可以说，符号串bb处于$T(M)$中；但我们现在感兴趣的是$bbabab$，由于

$\delta(q_0,\ a)=q_2$，所以$\delta(q_0,\ bba)=q_2$；又由于$\delta(q_2,\ b)=q_3$，所以$\delta(q_0,\ bbab)=q_3$；最后，由于$\delta(q_3,\ a)=q_1$及$\delta(q_1,\ b)=q_0$，所以$\delta(q_0,\ bbabab)=q_0$。可见，$bbabab$是$T(M)$中的符号串。易于看出，$T(M)$是语言$\{a,\ b\}^*$中的全部符号串的集合，$\{a,\ b\}^*$包括偶数个a及偶数个b。

符号串由奇数个a后面跟着奇数个b组成的语言，它的有限自动机如下：

$M=(K,\ \Sigma,\ \delta,\ q_0,\ F)$

$K=\{q_0,\ q_1,\ q_2\}$

$\Sigma=\{a,\ b\}$

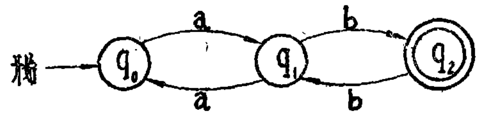
$q_0=\{q_0\}$

$F=\{q_2\}$

$\delta(q_0,\ a)=q_1$　　$\delta(q_1,\ b)=q_2$

$\delta(q_1,\ a)=q_0$　　$\delta(q_2,\ b)=q_1$

其状态图是：



如果把符号串$aaab$输入到M中，由于$\delta(q_0,\ a)=q_1$及$\delta(q_1,\ a)=q_0$，所以$\delta(q_0,\ aa)=q_0$；又由于$\delta(q_0,\ a)=q_1$，所以，$\delta(q_0,\ aaa)=q_1$；由于$\delta(q_1,\ b)=q_2$，所以，$\delta(q_0,\ aaab)=q_2$。可见，$aaab$可被这个自动机接收。

我们再来看可接收语言$\{a\ b^*\}$的自动机，这种语言的符号串由一个a后面跟着任意数目个b组成，或者符号串只有a。接收它的自动机如下：

$M=(K,\ \Sigma,\ \delta,\ q_0,\ F)$

$K=\{q_0,\ q_1,\ q_2\}$

$\Sigma=\{a,\ b\}$

$q_0=\{q_0\}$

$F=\{q_2\}$

$\delta(q_0,\ a)=\{q_1,\ q_2\}$

$\delta(q_0,\ b)=\phi$

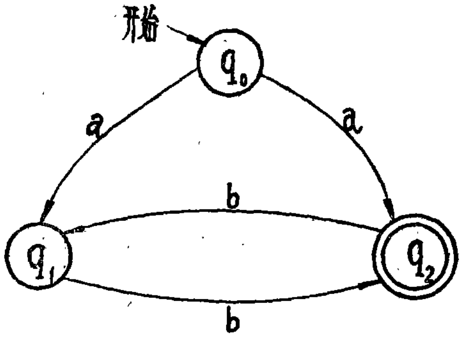
$\delta(q_1,\ a)=\phi$

$\delta(q_1,\ b)=q_2$

$\delta(q_2,\ a)=\phi$

$\delta(q_2,\ b)=q_1$

其状态图是：



在这种自动机中，如果M处于状态q_0并扫描a，它可以有状态q_1或状态q_2可供选择，这种可选择状态的自动机，叫做非确定的自动机，而前面那几个自动机不能选择状态，叫确定的自动机。二者区别在于，在非确定的自动机中，$\delta(q,\ a)$是一系列状态的集合，即$\delta(q,\ a)=\{p_1,\ p_2,\ \cdots,\ p_k\}$当$M$处于状态$q$并扫描输入带子上的$a$时，其读数头向右移动一个单元，并选择$p_1$、$p_2$，…，$p_k$中的任意一个状态作为其下一个状态。

M. O. Rabin和D Scott证明了，对于可被非确定的有限自动机接收的任何语言来说，存在着可接收同一语言的确定的有限自动机[6]。

在前面举出的有限自动机的例子中，读数头只能在一个方向上从左到右移动，这种有限自动机叫做单路有限自动机。还有一种有限自动机，它的读数头可以在两个方向上移动，也就是说，它可以重新扫描已经读过的符号，这种有限自动机叫做双路有限自动机[7]。用R表示向右移动一个单元，用L表示向左移动一个单元，用S表示不移动，如果选择D来代表这些符号，即$D=\{R、L，S\}$，那么，双路有限自动机的δ映射可表示为：

$\delta(q_i,\ a_j)=(q_k,\ D)$

当M处于状态q_i，扫描输入符号a_j时，根据

D到底是等于L、R或是S，将其读数头向左、向右移动一个单元，或者始终不移动其读数头，并把M的状态改变到q_k。

M. O. Rabin、D. Scott 和 J. C. Shepherdson 证明了，能被双路有限自动机接收的语言或集合的类，与能被单路有限自动机接收的语言或集合的类是相同的[6]、[7]。

2. 后进先出自动机

后进先出自动机是一种能控制输入带子和后进先出存储器的自动机。后进先出存储器就是一个后进先出的存储带子，符号的加进或擦去遵守着"后进先出"的原则[8]。

这种后进先出存储器可以同一种装盘子的栈相类比。在这种栈中，弹簧支在盘子的下边，它的力量刚刚能使得只有一个盘子出现在栈表面上。当栈顶的盘子被拿走时，弹簧的负荷减小，因而下面一个盘子在弹簧的推动下就马上出现在栈表面上；如果把一个盘子加到栈顶，这一堆盘子就被往下推，弹簧受到压缩，因而这个盘子就刚刚与栈表面相平。我们假定弹簧是任意长的，因此，我们想加多少盘子就可以把多少盘子加上去。

把这种装盘子的栈配上一个控制器来识别有中心元素的镜象结构语言 $L=\{aca^*\}$。控制器可处于两个状态 q_1 或 q_2，这种装盘子的栈相当于后进先出存储器，该存储器中的符号用兰、绿、红三色盘子表示，再配上输入带子，就是一个后进先出自动机。它操作如下。

① 自动机从栈上的一个红盘子开始工作，这时，控制器处于状态 q_1。

② 如果输入是a，该装置处于状态q_1，就把一个兰盘子放在栈上；如果输入是 b，该装置处于状态q_1，就把一个绿盘子放在栈上。这两种情况控制器仍保持状态q_1。

③ 如果输入是c，该装置处于状态 q_1，就把状态 q_1 变为 q_2，而不加上或去掉任何盘子。

④ 如果输入是a，该装置处于状态q_2，并且放在栈顶的是兰盘子，那么，就把这个盘子去掉；如果输入是 b，该装置处于状态 q_2，并且放在栈顶的是绿盘子，那么，就把这个盘子去掉。这两种情况，控制器仍保持状态q_2。

⑤ 如果该装置处于状态q_2，而放在栈顶的是红盘子，那么，就把这个盘子去掉，不必等待下一个输入。

⑥ 此外，不能作其它动作。

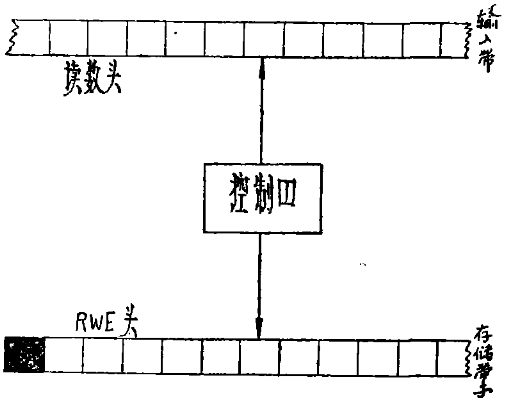
如果在处理输入符号串的最后一个符号时，栈完全变空，那么，就说，这种后进先出自动机接收了输入符号串。

具体地说，这种后进先出自动机接收语言 $L=\{aca^*\}$ 的操作过程如下：

在状态 q_1，当输入a时，把一个兰盘子放在栈顶，当输入b时，把一个绿盘子放在栈顶，当输入c时，该装置转到状态 q_2. 然后，在状态q_2，把输入符号串中的剩余部分同刚才存入栈中的盘子相对照，当这部分输入符号为a时，从栈顶去掉一个兰盘子，当这部分输入符号为b时，从栈顶去掉一个绿盘子。如果栈顶盘子颜色与这部分输入的符号不相配，那么，该装置就不能继续进行输入。如果栈顶盘子颜色与这部分输入的符号一个一个都相配，那么，位于栈最底部的红盘子就会突然地升到栈表面上来，马上去掉这个红盘子，装盘子的栈完全变空。这时，我们就可以说，这种后进先出自动机接收了语言 $L=\{aca^*\}$。

后进先出自动机由控制器、输入带子、后进先出存储带子，读数头及读—写—擦去头（Read—write—erase head，简写为 RWE 头）组成。控制器可有有限个状态，包括一个初始状态和若干个最后状态。带子被分为若干单元，每个单元只能记录一个符号。输入带子两端可任意延伸。存储带子的一端是有界的，另一端可无限延伸，最左端的符号被认为是处于后进先出存储器的顶端，越接近左端的符号在后进先出存储器中

的位置就越高。读数头每次只能从输入带子中读一个符号，RWE 头每次可以在存储带子上读、写或者擦去一个符号。其结构原理图示如下：



后进先出自动机的运动有两种类型：第一种类型的运动要扫描输入符号，根据输入符号、后进先出存储器顶端的符号以及控制器的状态三个因素来决定控制器的下一状态以及决定选择什么符号来替代后进先出存储器顶端的符号，在进行选择之后，就把读数头向前推进一个符号。第二种类型的运动叫做 ε 运动(ε-move)，这种运动不使用输入符号，而且在运动之后也不把读数头向前推进。ε 运动使得自动机可以不使用输入符号而对后进先出存储器进行操作。

一个后进先出自动机 M 可定义为七元组：

$$M=(K,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ Z_0,\ F)$$

这里，

① K 是状态的有限集合；

② Σ 是有限的输入字母表；

③ Γ 是有限的后进先出字母表；

④ K 中的 q_0 是初始状态；

⑤ Γ 中的 Z_0 是最先出现在后进先出存储器上的初始符号；

⑥ $F\subseteq K$ 是最后状态的集合；

⑦ δ 是 $K\times(\Sigma\cup\{\varepsilon\})\times\Gamma$ 到 $K\times\Gamma^*$ 的有限子集合的映射［Γ^* 表示由 Γ 中的符号构成的全部符号串（包括空符号串 ε）的集合］。

与上述两种运动类型相对应，δ 映射也有两种形式。

δ 映射的一种形式是：

$$\delta(q,a,Z)=\{(p_1,\gamma_1),\ (p_2,\gamma_2),\cdots,\ (p_m,\ \gamma_m)\}。$$

其中，q 与 $p_i(1\leqslant i\leqslant m)$ 在 K 中，a 在 Σ 中，Z 在 Γ 中，$\gamma_i(1\leqslant i\leqslant m)$ 在 Γ^* 中。这个式子的意思是：后进先出自动机在状态为 q，输入符号为 a，后进先出存储器顶端的符号为 Z 的情况下，把状态 q 变为状态 p_i，用 γ_i 代替 Z，并把读数头向前推进一个符号。如要擦去 Z，可令 $\gamma_i=\varepsilon$。

另一种 δ 映射是描写 ε 运动的，它的形式是：

$$\delta(q,\varepsilon,Z)=\{(p_1,\gamma_1),\ (p_2,\gamma_2),\cdots,\ (p_m,\ \gamma_m)\}。$$

这个式子的意思是：后进先出自动机在状态为 q，后进先出存储器顶端的符号为 Z，而且不管被扫描的输入符号是什么的情况下，把状态 q 变为状态 p_i，并对于任何的 $i(1\leqslant i\leqslant m)$，用 γ_i 替换 Z。在这种场合，读数头不向前推进。如要擦去 Z，可令 $\gamma_i=\varepsilon$。

例如，前述接收语言 $L=\{aca^*\}$ 的后进先出自动机 M 可写为：

$$M=(K,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ Z_0,\ F)$$

$K=\{q_1,\ q_2\}$

$\Sigma=\{a,\ b,\ c\}$

$\Gamma=\{R,\ B,\ G\}$，（R 表示红盘子，B 表示兰盘子，G 表示绿盘子）

$q_0=\{q_1\}$

$Z_0=\{R\}$，（R 表示红盘子）

$F=\{\phi\}$，（ϕ 表示空集合）

δ 映射如下：

$\delta(q_1,\ a,\ R)=\{(q_1,\ BR)\}$

$\delta(q_1,\ a,\ B)=\{(q_1,\ BB)\}$

$\delta(q_1,\ a,\ G)=\{(q_1,\ BG)\}$

$\delta(q_1,\ b,\ R)=\{(q_1,\ GR)\}$

$\delta(q_1,\ b,\ B)=\{(q_1,\ GB)\}$

$\delta(q_1,\ b,\ G)=\{(q_1,\ GG)\}$

$\delta(q_1, c, R)=\{(q_2, R)\}$

$\delta(q_1, c, B)=\{(q_2, B)\}$

$\delta(q_1, c, G)=\{(q_2, G)\}$

$\delta(q_2, a, B)=\{(q_2, \varepsilon)\}$

$\delta(q_2, b, G)=\{(q_2, \varepsilon)\}$

$\delta(q_2, \varepsilon, R)=\{(q_2, \varepsilon)\}$

注意，规则 $\delta(q_2, \varepsilon, R)=\{(q_2, \varepsilon)\}$ 的意思是：在状态为q_2，后进先出存储器顶端的符号为 R 时， 可擦去 R 而不管输入符号是什么。在这种场合，读数头不向前推进，自动机作的是 ε 运动。

显然，从其δ映射可看出，这是确定的后进先出自动机。

如果要接收没有中心元素的镜象结构语言 $L=\{\alpha\alpha^*\}$，可提出如下后进先出自动机 M：

$M=(K, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

$K=\{q_1, q_2\}$

$\Sigma=\{a, b\}$

$\Gamma=\{R, B, G\}$

$q_0=\{q_1\}$

$Z_0=\{R\}$

$F=\{\phi\}$

δ映射如下

$\delta(q_1, a, R)=\{(q_1, BR)\}$ (i)

$\delta(q_1, b, R)=\{(q_1, GR)\}$ (ii)

$\delta(q_1,a,B)=\{(q_1,BB),(q_2,\varepsilon)\}$ (iii)

$\delta(q_1, a, G)=\{(q_1, BG)\}$ (iv)

$\delta(q_1, b, B)=\{(q_1, GB)\}$ (v)

$\delta(q_1,b,G)=\{(q_1,GG),(q_2,\varepsilon)\}$ (vi)

$\delta(q_2, a, B)=\{(q_2, \varepsilon)\}$ (vii)

$\delta(q_2, b, G)=\{(q_2, \varepsilon)\}$ (viii)

$\delta(q_1, \varepsilon, R)=\{(q_2, \varepsilon)\}$ (ix)

$\delta(q_2, \varepsilon, R)=\{(q_2, \varepsilon)\}$ (x)

规则(i)—(vi)使 M 把输入符号串存储在后进先出存储器上，但其中的规则(iii)及规则(vi)，M必须对运动进行选择，“猜测”是否已达到输入符号串镜象结构的中点，如果 M 判断已达到了这个中点，那么，就选择（q_2，ε），并转入状态 q_2，然后，把输入符号串中剩下的部分同后进先出存储器中已存入的盘子相对比。要是 M 的判断是对的，那么，后进先出存储器顶端盘子的颜色必然与这一部分输入的符号一个一个都相配，M 可将后进先出存储器变空，从而接收语言 $\{\alpha\alpha^*\}$。要是 M 的判断是错的，那么，它就不能接收语言 $\{\alpha\alpha^*\}$。

显然，从其δ映射可看出，这是非确定的后进先出自动机。

后进先出自动机的格式(configuration)是由q及γ构成的对(q，γ)。其中，q是K中的状态，γ是由后进先出字母表中的符号构成的符号串。如果M处于状态q，γ处于后进先出存储器上，而且γ最左端的符号是后进先出存储器顶端的符号，那么就说，后进先出自动机M在格式（q，γ）中。如果a在$\Sigma\cup\{\varepsilon\}$中，$\gamma$及$\beta$在$\Gamma^*$中，$Z$在$\Gamma$中，对（$p$，$\beta$）在$\delta$（$q$，$a$，$Z$）中，那么就写为：

$$a\text{: } (q, Z\gamma) \underset{M}{\vdash} (p, \beta\gamma).$$

这个式子表示：根据后进先出自动机M的规则，输入符号a可使M从格式（q，$Z\gamma$）转到格式（p，$\beta\gamma$）。

如果对于$\Sigma\cup\{\varepsilon\}$中的每一个$a_1, a_2, \cdots, a_n$，状态$q_1$，$q_2$，…，$q_{n+1}$以及由后进先出字母表中的符号构成的符号串$\gamma_1$，$\gamma_2$，…，$\gamma_{n+1}$，对于1与$n$之间的一切$i$，有

$$a_i\text{: } (q_i, \gamma_i) \underset{M}{\vdash} (q_{i+1}, \gamma_{i+1}),$$

那么，就写为：

$$a_1a_2\cdots a_n\text{: } (q_1,\gamma_1) \underset{M}{\overset{*}{\vdash}} (q_{n+1}, \gamma_{n+1}).$$

现在定义被后进先出自动机接收的语言。有两种方法：

第一种是把被后进先出自动机接收的语言定义为全部输入符号串的集合，对于这个集合，作一系列的运动使后进先出自动机把它的后进先出存储器变空，这种语言可看成

是以空存储器接收的语言，记为$N(M)$。这样，

$$N(M)=\{W|W: \text{对于}K\text{中的任何的}q,\ \text{有}(q_0,Z_0)\overset{*}{\underset{M}{\vdash}}(q,\ \varepsilon)\}.$$

第二种同有限自动机接收输入带子的方法相似。把K中的某些状态记为最后状态，并把被后进先出自动机接收的语言定义为全部输入符号串的集合，作一系列的运动使后进先出自动机进入最后状态。这种语言，可看成是以最后状态接收的语言，记为$T(M)$。这样，

$$T(M)=\{W|W: \text{对于}\Gamma^*\text{中任何的}\gamma\ \text{及}F\text{中的}q,\ (q_0,\ Z_0)\overset{*}{\underset{M}{\vdash}}(q,\ \gamma)\}.$$

N Chomsky 证明了，对于某一个后进先出自动机M_1，L是$N(M_1)$，当且仅当对于另一个后进先出自动机M_2，L是$T(M_2)$。也就是说，如果某一语言L能被某一后进先出自动机M_1以空存储器接收，那么，它也可以被另一个后进先出自动机M_2以最后状态接收[10]。

N. Chomsky 还证明了，如果L是上下文无关语言，那么，存在着一个后进先出自动机M，使得$L=N(M)$；反之，如果对于某一后进先出自动机M，$L=N(M)$，那么，L是上下文无关语言[10]。

由此可见，下面三种陈述方式是等价的：

① L是上下文无关语言。

② 对于某一后进先出自动机M_1，$L=N(M_1)$。

③ 对于某一后进先出自动机M_2，$L=T(M_2)$。

因此可以说，后进先出自动机是上下文无关语言的接收机[10]、[12]、[14]。不过，这种说法不是很严格的。人们通过进一步的研究发现，对于非确定的后进先出自动机来说，它接收的全部都是上下文无关语言，而对于确定的后进先出自动机来说，它接收的却只是上下文无关语言中的一部分，这一部分叫做确定语言（deterministic language）[17]、[19]。确定语言可用$LR(K)$文法来孳生。由于$LR(K)$文法是一种非二义性的上下文无关文法，因此，这种文法是研究上下文无关语言的一种非常有用的工具注。

3. 图灵机

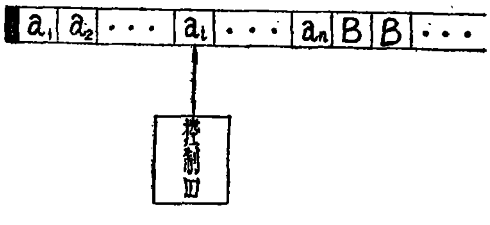
最基本的图灵机由控制器、带子和读写头组成。控制器有有限个状态，带子的左端是有界的，它可以从最左端起记录输入符号，但其右端则是无限长的。带子分为许多单元，输入符号记录在带子靠左边的单元上，每个单元记录一个符号，而其余的单元则记录着空白，空白用B表示。读写头既可扫描符号，也可写下符号。根据读写头所扫描的符号及控制器所处的状态，图灵机可以作如下运动：

①、改变状态；

②、在读写头所扫描的带子的单元上写下非空白符号来代替该单元上原有的符号；

③、使读写头向左或向右移动一个单元。读写头的运动用D表示，$D=\{L, R\}$，其中L表示向左运动，R表示向右运动。

其结构原理图示面下：



一个图灵机T可定义为六元组

$$T=(K,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ F)$$

其中，

K是状态的有限集合；

注 关于$LR(K)$文法，可参看：王翰虎、$LR(K)$文法介绍，《计算机应用与应用数学》，1976年第8期，第30—44页。

Σ是输入字母表；

Γ是可在带子上记录的符号的有限集合，包括空白符号B，因此，$\Sigma \subseteq \Gamma$；

q_0是K中的初始状态；

$F \subseteq K$是最后状态的集合；

δ是$K \times \Gamma$到$K \times (\Gamma - \{B\}) \times D$的映射。

其一般形式为

$\delta(q_i, a) = (q_j, b, D)$.

它的意思是：图灵机在状态为q_i，扫描符号a时，把状态q_i变为q_j，在符号a的单元处写下b以代替a，并向左或向右移动一个单元。

图灵机的格式是三元组(q, α, i)，其中，q是K中的状态，α是$(\Gamma - \{B\})^*$中的符号注，是带子的非空白部分，α的右边是无限个空白符号，i是一个整数，它表示读写头到α左端的距离。

图灵机T的运动如下：

设$(q, A_1A_2\cdots A_n, i)$是T的一个格式，这里，$1 \leqslant i \leqslant n+1$.

若$1 \leqslant i \leqslant n$，且$\delta(q, A_i) = (p, A, R)$，则记为：

$$(q, A_1A_2\cdots A_n, i) \underset{T}{\vdash} (p, A_1A_2\cdots A_{i-1}AA_{i+1}\cdots A_n, i+1)$$

即，T写下符号A并向右移动一个单元。

若$2 \leqslant i \leqslant n$，且$\delta(q, A_i) = (p, A, L)$，则记为：

$$(q, A_1A_2\cdots A_n, i) \underset{T}{\vdash} (p, A_1A_2\cdots A_{i-1}AA_{i+1}\cdots A_n, i-1)$$

即，T写下A并向左移动一个单元，但是，不能超出带子左端的界限。

当$i = n+1$时，读写头扫描空白B. 这时，若$\delta(q, B) = (p, A, R)$，则记为：

$$(q, A_1A_2\cdots A_n, n+1) \underset{T}{\vdash} (p, A_1A_2\cdots A_nA, n+2)$$

若换成$\delta(q, B) = (p, A, L)$，则记为：

$$(q, A_1A_2\cdots A_n, n+1) \underset{T}{\vdash} (p, A_1A_2\cdots A_nA, n)$$

若两个格式以$\underset{T}{\vdash}$相联系，就说，第二个格式是第一个格式通过一次运动而来的结果。如果一个格式是由另一个格式通过有限次运动而来的结果，它们之间就用关系符号$\overset{*}{\underset{T}{\vdash}}$连接。

被图灵机T接收的语言L是Σ^*中这样的符号串w，如果把该符号串w记录在T的带子上，T能从控制器的状态为q_0、读写头处于最左端开始，经过有限次运动，把输入符号串w变成Γ^*中的符号串α，并进入F中的某个最后状态q。因此，被T接收的语言L可表示为：

$L = \{w \mid w$在Σ^*中，并且对于F中的某个q，Γ^*中的α及整数i，有$(q_0, w, 1) \overset{*}{\underset{T}{\vdash}} (q, \alpha, i)\}$.

下面，我们给出接收上下文无关语言$L = \{a^nb^n\}$的图灵机T。

$T = (K, \Sigma, \Gamma, \delta, q_0, F)$

其中，$K = \{q_0, q_1, q_2, q_3, q_4, q_5\}$

$\Sigma = \{a, b\}$

$\Gamma = \{a, b, B, X, Y\}$

$q_0 = \{q_0\}$

$F = \{q_5\}$

δ映射如下：

$\delta(q_0, a) = (q_1, X, R)$ (i)

$\delta(q_1, a) = (q_1, a, R)$ (ii)

$\delta(q_1, Y) = (q_1, Y, R)$ (iii)

$\delta(q_1, b) = (q_2, Y, L)$ (iv)

$\delta(q_2, Y) = (q_2, Y, L)$ (v)

$\delta(q_2, X) = (q_3, X, R)$ (vi)

$\delta(q_2, a) = (q_4, a, L)$ (vii)

$\delta(q_4, a) = (q_4, a, L)$ (viii)

注 $(\Gamma - \{B\})^*$表示由$(\Gamma - \{B\})$中的符号构成的全部符号串（包括空符号串）的集合。Σ^*意义亦同。

$\delta(q_4, X) = (q_0,\ X,\ R)$ (ix)

$\delta(q_3, Y) = (q_3,\ Y,\ R)$ (x)

$\delta(q_3, B) = (q_5,\ Y,\ R)$ (xi)

例如，图灵机T接收符号串 aaabbb 的运动过程，用相互连续的格式表示如下，每个格式后面记下所用规则的号码，箭头表示读写头所扫描的符号：

格式	所用规则
(q_0, aaabbb, 1)	开始
(q_1, Xaabbb, 2)	(i)
(q_1, Xaabbb, 3)	(ii)
(q_1, Xaabbb, 4)	(ii)
(q_2, XaaYbb, 3)	(iv)
(q_4, XaaYbb, 2)	(vii)
(q_4, XaaYbb, 1)	(viii)
(q_0, XaaYbb, 2)	(ix)
(q_1, XXaYbb, 3)	(i)
(q_1, XXaYbb, 4)	(ii)
(q_1, XXaYbb, 5)	(iii)
(q_4, XXaYYb, 4)	(iv)
(q_2, XXaYYb, 3)	(v)
(q_4, XXaYYb, 2)	(vii)
(q_0, XXaYYb, 3)	(ix)
(q_1, XXXYYb, 4)	(i)
(q_1, XXXYYb, 5)	(iii)
(q_1, XXXYYb, 6)	(iii)
(q_2, XXXYYY, 5)	(iv)
(q_2, XXXYYY, 4)	(v)
(q_2, XXXYYY, 3)	(v)
(q_3, XXXYYY, 4)	(vi)
(q_3, XXXYYY, 5)	(x)
(q_3, XXXYYY, 6)	(x)
(q_3, XXXYYY, 7)	(x)
(q_5, XXXYYYY, 8)	(xi)

待接收的符号串 aaabbb 记录在带子上，T 从状态 q_0 开始，并扫描带子上最左边的符号，如果这个符号是 a，读写头就在上面写 X，用 X代替 a，并向右移动一个单元，这时，T的状态改变到 q_1。读写头继续向右移动，逐一扫描遇到的 a 以查找 b，当查找到 b 时，状态由q_1变为q_2，读写头在上面写 Y，用 Y 代替 b，并向左移动一个单元，这个过程继续进行，交替地把一个 a 转变为X，把一个 b 转变为Y，当读写头扫描到空白时，T就进入最后状态 q_5. 这时，我们就说，输入符号串 aaabbb 作为成立句子被图灵机接收了。

上面所讨论的最基本的图灵机可以通过“扩充”和“限制”加以变换，从而得到各种类型的图灵机。在“扩充”这一方面，我们可以去掉输入带子左端有界这样的规定，而使带子在左右两端都可以无限延伸，或者使带子在四个方向甚至 n 个方向上都可以无限延伸，我们还可以把带子扩充为一个以上，把读写头也扩充为一个以上等等。在“限制”这一方面，我们可以把带子限制为只读带子，并可限制T的状态数和输入字母表中符号数。已经证明，通过这些变换后的图灵机与最基本的图灵机是等价的，这些变换并没有改变最基本的图灵机的计算能力[23]、[29]。

如果我们能够把某一个图灵机的操作指令在另一个图灵机的带子上编码，那么，第二个图灵机显然就能模拟第一个图灵机。一般地说，如果我们能够把任意的图灵机T的操作指令在另一个图灵机 U 的带子上编码，那么，我们就能够描写可以模拟任何任意的图灵机 T 的图灵机 U。U 叫做通用图灵机(universal Turing machine)。我们可以

把通用图灵机看成是一个通用计算机，这种通用计算机可以模拟包括它本身在内的任何一种计算机[1]。

N·Chomsky 证明了，如果语言 L 是由 0 型文法孳生的，那么，L 就能被图灵机识别；反之，如果语言 L 是由图灵机识别的，那么，L 就能被 0 型文法孳生 [12]。因此，图灵机是 0 型语言的接收机。

4. 线性有界自动机

线性有界自动机是有如下限制的图灵机，其限制是：带子上的输入符号串是左右都有界的符号，读写头在其向左或向右的运动中，任何时候都绝对不能超出这个界限。

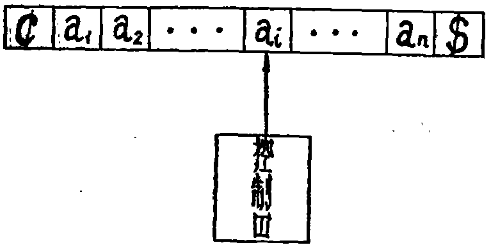
P. S. Landweber 和 S. Y. Kuroda 分别证明了，如果 L 是上下文有关语言，那么，L 就可被线性有界自动机接收；反之，如果 L 可被线性有界自动机接收，那么，L 就是上下文有关语言。因此，上下文有关语言的接收机是线性有界自动机 [15]、[16]。

一个线性有界自动机 M 可定义为六元组：

$$M=(K, \Sigma, \Gamma, \delta, q_0, F)$$

这些符号表示的意思与图灵机中基本上一样。需要说明的是，Σ 还包括两个特殊符号 $¢$ 和 $\$$；$¢$ 是左端标示符，$\$$ 是右端标示符，它们处于输入符号串的两端，以使读写头不能超出开始记录在带子上的输入符号串的范围。

线性有界自动机的结构原理图示如下：



线性有界自动机 M 的格式记为

$$(q, A_1A_2\cdots A_n, i),$$

这里，q 在 K 中，$A_1A_2\cdots A_n$ 在 Γ 中，i 是 1 与 n 之间的整数。

如果 $\delta(q, A_i)$ 包含 (p, A, L)，且 $i>1$，就说，

$$(q, A_1A_2\cdots A_n, i)\vdash_M (p, A_1A_2\cdots A_{i-1}AA_{i+1}\cdots A_n, i-1).$$

如果 $\delta(q,A_i)$ 包含 (p,A,R)，且 $i<n$，就说，

$$(q, A_1A_2\cdots A_n, i)\vdash_M (p, A_1A_2\cdots A_{i-1}AA_{i+1}\cdots A_n, i+1).$$

即，当 M 处于状态 q 并扫描 A_i 时，在 A_i 处写下 A 并代替 A_i，状态 q 改变到状态 p，并把其读写头向左或向右移动一个单元，但不能超出符号串原来出现的范围。

如果

$$(q_1, \alpha_1, i_1)\vdash_M (q_2, \alpha_2, i_2)$$

且

$$(q_2, \alpha_2, i_2)\vdash_M (q_3, \alpha_3, i_3),$$

那么，

$$(q_1, \alpha_1, i_1)\overset{*}{\vdash_M} (q_3, \alpha_3, i_3).$$

这样一来，被线性有界自动机 M 接收的上下文有关语言 L 可表示为：

$$L=\{w \mid w \text{ 在 } (\Sigma-\{¢, \$\})^* \text{ 中，并且对于 } F \text{ 中的某个 } q\text{，}\Gamma^* \text{ 中的 } \alpha \text{ 及整数 } i\text{，有 } (q_0, ¢w\$, 1)\overset{*}{\vdash_M}(q, \alpha, i)\}.$$

下面，给出能接收上下文有关语言 $L=\{a^nb^nc^n\}$ 的线性有界自动机 M：

$$M=(K, \Sigma, \Gamma, \delta, q_0, F)$$
$$K=\{q_0, q_1, q_2, q_3, q_4\}$$
$$\Sigma=\{a, b, c, ¢, \$\}$$
$$\Gamma=\{a, b, c, ¢, \$, X\}$$

$q_0=\{q_0\}$

$F=\{q_4\}$

δ 映射如下：

$\delta(q_0, a)=(q_1, X, R)$

$\delta(q_0, X)=(q_0, X, R)$

$\delta(q_0, \$)=(q_4, \$, S)$

（S 表示读写头不移动）

$\delta(q_1, a)=(q_1, a, R)$

$\delta(q_1, b)=(q_2, X, R)$

$\delta(q_1, X)=(q_1, X, R)$

$\delta(q_2, b)=(q_2, b, R)$

$\delta(q_2, c)=(q_3, X, L)$

$\delta(q_2, X)=(q_2, X, R)$

$\delta(q_3, X)=(q_3, X, L)$

$\delta(q_3, a)=(q_3, a, L)$

$\delta(q_3, b)=(q_3, b, L)$

$\delta(q_3, ¢)=(q_0, ¢, R)$

待接收的符号串 $\{a^nb^nc^n\}$ 记录在带子上，M从状态q_0开始，找到a，把a变为X，把读写头向右移动一个单元，并把状态 q_0 改变到状态 q_1。在状态 q_1，读写头继续向右运动，逐一扫描遇到的 a 以查找 b，如果 M 在状态 q_1 查找到 b，就把 b 变为 X，把读写头向右移动一个单元，并把状态 q_1 改变到 q_2。在状态 q_2，读写头继续向右运动，逐一扫描遇到的 b 以查找c，如果 M 在状态 q_2 查找到 c，就把 c 变为 X，把读写头向左移动一个单元，把状态 q_2 改变到 q_3。接着 M 在状态 q_3 从右到左逐一扫描它刚才扫描过的各个符号，一直扫描到左端标示符 ¢，并把状态 q_3 改变为状态 q_0。然后重复进行上述过程。如果 M 在状态 q_0 达到了右端标示符 \$，那么，读写头不再移动，M 停止。这时就说，符号串 $\{a^nb^nc^n\}$ 作为成立句子被 M 接收了。

本文所论的文法和自动机可归纳如下表：

语言和文法的名称	加在规则 P 上的限制	自动机
0 型	$\varphi\to\psi$, $\varphi\neq\phi$	图灵机
1型(上下文有关)	$\varphi\to\psi$, $\lvert\psi\rvert\geqslant\lvert\varphi\rvert$	线性有界自动机
2型(上下文无关)	$A\to\omega$, $A\in V_N$, $\omega\in V$	后进先出自动机
3型（有限状态）	$A\to aQ$, $A\to a$, $A, Q\in V_N$, $a\in V_T$	有限自动机

程序语言的内部结构，可以用形式语言的文法来加以描写。

例如，程序语言ALGOL中使用 Backus 范式表示赋值语句：

〈赋值语 句〉::=〈变 量〉=〈表 达 式〉

这种赋值语句相当于上下文无关文法的如下重写规则：

$$ASS \longrightarrow VAR \quad EQ \quad EXP$$

其中，ASS 表示赋值语句，VAR 表示变量，EQ 表示等号，EXP 表示表达式。

计算机的编译程序是把用 ALGOL、FORTRAN 或 COBOL 这一类程序语言写的程序作为数据接收，并把它们翻译为机器指令程序，以便在计算机上进行运算。因而在某种意义上，我们可以把编译程序看成是 ALGOL 等程序语言的接收程序，并把这种编译程序从形式上描写为一个自动机。

因此形式语言理论对程序语言和编译程序的研究有着重要意义[27]、[29]。同时对机器翻译和情报检索等信息加工工作，也是很有帮助的[8]、[28]、[30]。

参 考 文 献

[1] A. M. Turing, Proc. London Math. Soc., 2—42, p. 230—265 (1936); 43, p. 544—546, (1937)

[2] N. Chomsky, IRE. Trans. on Inform. Theory, IT—2(3), p. 113—124, (1956)
[3] N. Chomsky, Syntactic Structures, Mouton & Co., The Hague, (1957)
[4] N. Chomsky, G. A. Miller, Inf. and Control, 1:2, p. 91—112, (1958)
[5] N. Chomsky, 同上, 2:2, p. 137—167, (1959)
[6] M. O. Rabin, D. Scott, IBM. J. Res., 3:2, p. 115—125, (1959)
[7] J. C. Shepherdson, 同上, 3:2, p. 198—200, (1959)
[8] A. G. Oettinger, Automatic syntactic analysis and the pushdown store, Proc. Symp. Applied Math., 12. American Mathematical Society, Providence, Rhode Island, (1961)
[9] A. Gill, Introduction to the Theory of Finite-state Machines, McGraw-Hill, New York, (1962)
[10] N. Chomsky, Context-free grammars and pushdown Storage, Quart. Prog. Dept, No. 65, MIT Res. Lab. Elect., p. 187—194, (1962)
[11] N. Chomsky, G. A. Miller, Introduction to the formal analysis of natural languages, Handbook of Mathematical Psychology, Vol. 2, New York, Wiley, p. 269—322, (1963)
[12] N. Chomsky, Formal properties of grammars, 同上, p. 323—418, (1963)
[13] N. Chomsky, M. P. Schützenberger, The algebraic theory of context-free languages, Computer Programming and Formal System , North Holland, Amsterdam, p. 118—161, (1963)
[14] M. P. Schützenberger, Inf. and Control, 6:3, p. 246—264, (1963)
[15] P. S. Landweber, 同上, 6:2, p. 131—136, (1963)
[16] S. Y. Kuroda, 同上, 7:2, p. 207—223 (1964)
[17] L. H. Haines, Generation and recognition of formal languages, Doctoral Thesis, MIT, Cambridge, Massachusetts, (1965)
[18] S. A. Greibach, JACM, 12:1, p. 42—52, (1965)
[19] S. Ginsburg, Inf. and Control, 9:6, p. 620—648, (1966)
[20] S. Ginsburg, The Mathematical Theory of Context-Free Languages, McGraw-Hill, New York, (1966)
[21] R. J. Nelson, Introduction to Automata, Wiley, New York, (1968)
[22] J. E. Hopcroft, J. D. Ullman, Formal Languages and their Relation to Automata, Addison-Wesley, Reading, Massachusetts, (1969)
[23] M. A. Arbib, Theories of Abstract Automata, Prentice-Hall, INC, (1969)
[24] R. M. Karp, Automata Theory, Foundations of Information Systems Enginearing, (1970)
[25] M. A. Harrison, Characterizations of Languages by Grammars and Automata, 同上, (1970)
[26] A. T. Berztiss, Data Structures, Theory and Practice, Academic Press, (1971)
[27] 山崎利治，菊池光昭，染谷诚，プログラムの理论，《総合コンピュータ辞典》(山下英男监修)，p. 323—361，(1972)
[28] R. Rustin, Ed., Natural Language Processing, Algorithmic Press, New York, (1973)
[29] J. A. Moyne, Advances in Information System Science, Vol. 5, p. 263——333, (1974)
[30] J. A. Moyne, International Journal of Computer and Information Sciences, Vol. 4, №3, p. 265—279 (1975).