# [HIT-FLAA]哈工大2020春形式语言与自动机复习笔记



# 有穷自动机 (Finite Automata)

## 1. 确定的有穷自动机 (DFA)

### 1. 形式化定义

**确定的有穷自动机 (Deterministic Finite Automata)**：DFA是一个五元组，如：$M=(Q,\ \Sigma,\ \delta,\ q_0,\ F)$，其中，

- $Q$ 是有限的状态集，包含DFA中所有的状态；
- $\Sigma$ 是有限的输入字符集，也就是DFA的字母表；
- $q_0$ 是初始状态，并且 $q_0 \in Q$；
- $F$ 是终结状态的集合，并且 $F \in Q$；
- $\delta$ 是状态转移函数，它是一个映射：$Q \times \Sigma \to Q$

> $\delta$ 要对 $Q$ 中所有的状态和 $\Sigma$ 中所有的状态的组合都要有定义，也就是笛卡尔积是一个单射

例：以自动门为例，使用 $0$ 表示关门信号，$1$ 表示开门信号，$p$ 表示关门状态，$q$ 表示开门状态，则DFA如下：

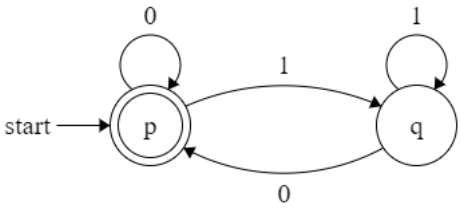
- 状态集：$\{p,\ q\}$
- 输入字符集：$\{0,\ 1\}$
- 初始状态：$p$

- 终结状态：$p$
- 状态转移函数：$\delta$

$$\delta(p,\ 0) = p$$
$$\delta(p,\ 1) = q$$
$$\delta(q,\ 1) = q$$
$$\delta(q,\ 0) = p$$

故该DFA的定义为：$\{\{p,\ q\},\ \{0,\ 1\},\ \delta,\ \ p,\ \{p\}\}$

**用图表示：**单线圈表示普通状态，双线圈表示终结状态，弧表示状态转移函数




**用表格表示：**用 $\rightarrow$ 标识出初始状态，用 $*$ 标识出终结状态

| | **0** | **1** |
|---|---|---|
| $\rightarrow *p$ | $p$ | $q$ |
| $q$ | $p$ | $q$ |

DFA的目的是区分字符串，于是，按照如上例子构造的DFA就把字符串分成了两类：

$L_1 = \{w \in \{0,1\}^* \mid w\ end\ with\ 0\} \cup \{\varepsilon\}$
$L_2 = \{w \in \{0,1\}^* \mid w\ end\ with\ 1\}$

而 $L_1$ 就是该DFA所接受的字符串集合，就能够判断任意的字符串 $w$，$w \in L_1$ 是否成立。

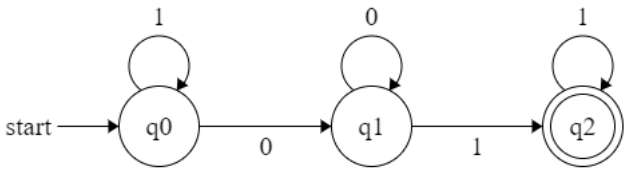
## 2. 构造

构造一个DFA的一般步骤：

- $w \in L$ 指的是哪些 $w$
- 根据能够产生的字符串划分等价类
- 根据划分的等价类设置状态
- 添加状态转移

例：构造DFA接受 $L = \{x01y \mid x\ and\ y\ are\ consists\ of\ any\ number\ of\ 0's\ and\ 1's\}$

- {ε,0,1,00,01,10,11,000,001,010,100,011,101,110,111,0000,0001,0010,0100,1000,0011,0101,1001, ...}
  高亮的是 $L$ 应该接受的字符串
- 观察这些字符串，有如下发现：
  没有0出现的时候，也就是都是1的时候效果是同样的，所以可以划分为一个类；
  有0出现，但没有1出现的时候等待1出现就可以了，所以这不同于上一类，又可以划分为一类；
  有0出现，有1出现，就有了01子串，而之后无论再出现什么都是会被接受的了，所以这是一类；
- 根据上面划分的3类可以设置三个状态 $q0$，$q1$，$q2$
- 添加状态转移得到结果：




**定义**：若 $A = (Q, \Sigma, \delta, q_0, F)$ 是一个 DFA，则 D 接受的语言为 $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$。

## 3. 正则语言

**定义**：如果存在一个DFA接受 $L$，那么就称 $L$ 是一个正则语言。这类 $L$ 就被称为正则语言。

> **练习**：Construct DFA for following languages :
>
> a) $\{0\}^*$
>
> b) $\{w \mid w \in 0,1^*\ and\ begin\ with\ 0\}$
>
> c) $\{w \mid w\ consists\ of\ any\ number\ of\ 0's\ followed\ by\ any\ number\ of\ 1's\}$
>
> d) $\{\varepsilon\}$

> e) $\phi$

## 2. 非确定的有穷自动机 (NFA)

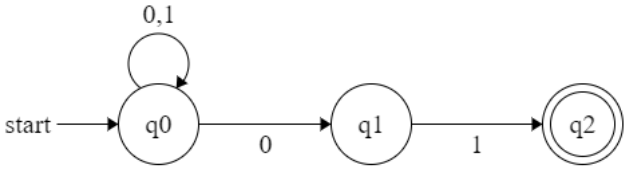
**形式化定义**

**非确定的有穷自动机 (Nondeterministic finite automaton)**：NFA是一个五元组，如：$M = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$，其中，

- $Q$ 是有限的状态集，包含NFA中所有的状态；
- $\Sigma$ 是有限的输入字符集，也就是NFA的字母表；
- $q_0$ 是初始状态，并且 $q_0 \in Q$；
- $F$ 是终结状态的集合，并且 $F \in Q$；
- $\delta$ 是状态转移函数，它是一个映射：$Q \times \Sigma \to 2^Q$

> 与DFA唯一的不同就是 $\delta$ 的象集是 $Q$ 的幂集，结果是一个集合

例：构造NFA接受 $L_{x01} = \{x01 \mid x\ is\ any\ strings\ of\ 0's\ and\ 1's\}$




可以看到，$\delta(q0,\ 0) = \{q0,\ q1\}$，得到的就是一个集合。

$\delta(q1,\ 0) = \phi$ 表明NFA不接受这个输入，NFA可以简化这种记法，但DFA不行。

**定义**：若 $A = (Q, \Sigma, \delta, q_0, F)$ 是一个 NFA，则 D 接受的语言为 $L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \phi\}$。

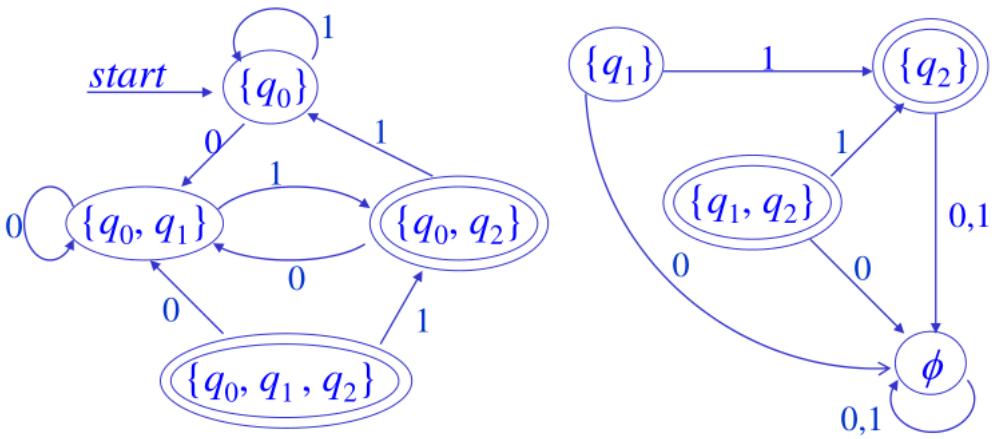
> 只需要有一条路能够让 $w$ 从 $q_0$ 走到终结状态就可以说NFA接受 $w$。这也是为什么它是非确定的。

## 3. DFA和NFA的等价性

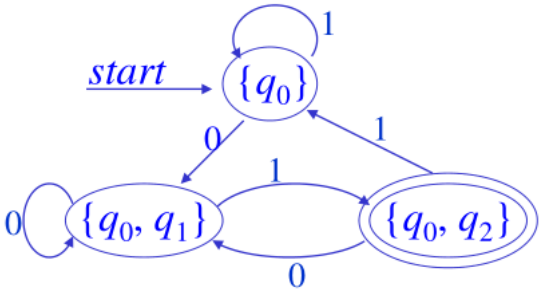
如果一个DFA和一个NFA接受的是同一个语言，那么就称这两个FA是等价的。而对于能构造一个DFA来接受它的语言来说，也必定能构造一个NFA来接受它，反之亦然。所以，所有的DFA和对应的NFA都是等价的。

> 证明：
>
> - 显然，如果有一个DFA接受L，则必定有一个NFA接受L；
>
>  给定一个DFA：$A = (Q_D,\ \Sigma,\ \delta_D,\ q_0,\ F_D)$，构造一个对应的NFA：$B = (Q_N,\ \Sigma,\ \delta_N,\ q_0,\ F_N)$
>
>  则 $Q_N = Q_D$，$\delta_N(q, a) = \{\delta_D(q, a)\}$，$F_N = F_D$。
> - 再证，如果有一个NFA接受L，则必定有一个DFA接受L。
>
>  给定一个NFA：$A = (Q_N,\ \Sigma,\ \delta_N,\ q_0,\ F_N)$，构造一个对应的DFA：$B = (Q_D,\ \Sigma,\ \delta_D,\ q_0,\ F_D)$
>
>  令 $Q_D = 2^{Q_N} = \{S \mid S \subseteq Q_N\}$
>
>  则 $\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$，因为$S$是 $Q_N$ 的一个子集，所以把 $S$ 中的每个元素在 $a$ 下确定状态后合并即可。
>
>  则 $F_D = \{S \mid S \subseteq Q_N\ and\ S \cap F_N \neq \phi\}$。*注：显然 $F_D$ 有可能不仅有一个元素。*

例：用上一节 (2) 中 $L_{x01}$ 的例子来看，NFA已经构造好了，用已有的NFA构造DFA如下：

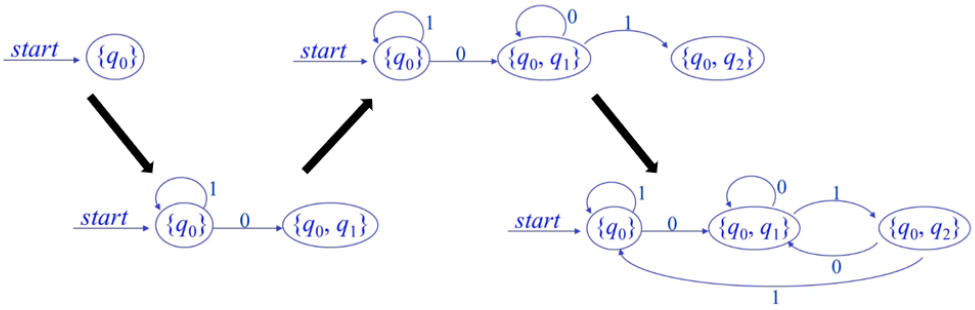



图中可以看出左半部分的 $\{q_0,\ q_1,\ q_2\}$ 这个状态显然是不可达的，**没有意义，可以删去**，右半部分同理也可删去。就简化成了如下的样子：




转化的另一个办法，**子集构造法**(Sub-set construction)，**惰性计算**，走一步看一步，较上面的办法清爽许多。基本过程是从初始状态开始，看它可能走到哪些状态，然后看它走到的状态又分别能走到哪些状态，如此循环直到没有新的状态出现。

还是用上面的例子做说明：




> **解题方法**：因为NFA的行为更接近于人的思维，所以构造DFA的题可以先构造NFA然后转化成DFA

# 4. ε-NFA和最小化DFA

## 1. ε-NFA的形式化定义

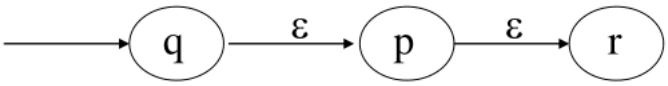
**带有空转移的非确定有穷自动机**：ε-NFA是一个五元组，如：$M = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$，其中，

- $Q$ 是有限的状态集，包含NFA中所有的状态；
- $\Sigma$ 是有限的输入字符集，也就是NFA的字母表；
- $q_0$ 是初始状态，并且 $q_0 \in Q$；
- $F$ 是终结状态的集合，并且 $F \in Q$；
- $\delta$ 是状态转移函数，它是一个映射：$Q \times \{\Sigma \cup \{\varepsilon\}\} \to 2^Q$

> 与NFA唯一的不同在于 $\delta$ 多了一个 $\varepsilon$ 输入，因而多了一个空转移行为。

## 2. ε-闭包

**ε-闭包**：状态q可以通过 (**一次或多次**) ε空转移到达的状态构成的集合就是q的ε-闭包，记为ECLOSE(q)，或更简单的E(q)。**自己到自己也是空转移！！！** 例：




$$E(r) = \{ r \},\ E(p) = \{ p, r \},\ E(q) = \{ p, q, r \}$$

在状态 $q$ 读了字符串 $w$ 可以到达的状态：

- NFA：$\hat{\delta}(q,\ w) = \{ p_1,\ p_2,\ \ldots,\ p_k \}$
- ε-NFA：$\hat{\delta}(q,\ w) = \bigcup_{i=1}^{m} E(r_i)$

**定义**：若 $A = (Q, \Sigma, \delta, q_0, F)$ 是一个 ε-NFA，则 D 接受的语言为 $L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \phi\}$。

> 其实和NFA接受的语言是一样的

## 3. DFA的最小化问题

> 最小化DFA就是找到一个等价的状态数最少的DFA

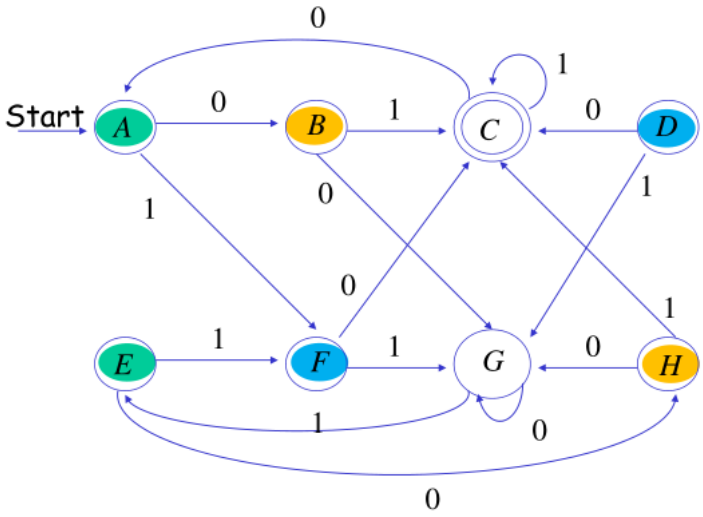
对于两个状态，一定是 **等价 / 可区分** 的。

- 等价：$\forall w \in \Sigma^*,\ \hat{\delta}(p,\ w) \in F \Leftrightarrow \hat{\delta}(q,\ w) \in F$

  **注意**：表明的是对于一个输入，两个状态都转移到终结状态或都转移到非终结状态，**并不一定相同！**
- 可区分：$\exists w \in \Sigma^*,\ \hat{\delta}(p,\ w) \in F \Leftrightarrow \neg\hat{\delta}(q,\ w) \in F$

  当两个状态为可区分时，存在至少一个输入符，转移后状态不都为终结状态或不都为非终结状态。

例：

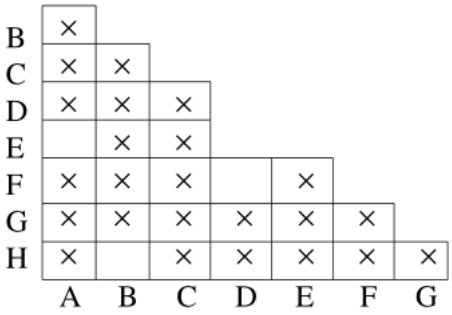



**------ Table-filling algorithm -------**

把所有的状态对画成一张表，逐个检查所有的状态对，如果可区分，则把该格子标记，直到填完，剩下的没有标记的格子就是不可区分（等价）的状态对。

> 判断两个状态对可区分的策略：
>
> - 终结状态和非终结状态一定是可区分的
> - 两个状态读相同的字符，一个到了终结状态，一个到了非终结状态，则是可区分的
>
> 所以，要找让一个状态到终结状态的输入串，看在这个串下另一个状态是不是到了非终结状态，如果是就能很快判断了。

上例用 Table-filling algorithm 得到的结果：




最后把等价的状态捏在一起就好了，就得到了最小化后的DFA。

---

# 正则表达式 (Regular Expression)

## 1. 正则表达式

正则表达式的递归定义：

- $\varepsilon$ 是一个正则表达式，表示语言 $\{\varepsilon\}$；
- $\phi$ 是一个正则表达式，表示空语言 $\phi$；
- $\forall a \in \Sigma$，$a$ 是一个正则表达式，表示语言 $\{a\}$；
- 如果 $E$ 和 $F$ 是正则表达式，表示的语言分别是 $L(E)$ 和 $L(F)$，则 $E+F$，$EF$，$E^*$ 都是正则表达式，分别表示的语言是 $L(E) \cup L(F)$，$L(E)L(F)$，$(L(E))^*$；
- 如果 $E$ 是正则表达式，则 $(E)$ 也是。

> - 运算符优先级：括号 > 星 > 连接 > 加
> - 每一个正则表达式都对应一个正则语言。

例：$L = \{w \mid w \in 0,1^* \ and \ w \ has \ no \ pair \ of \ consecutive \ 0's\}$

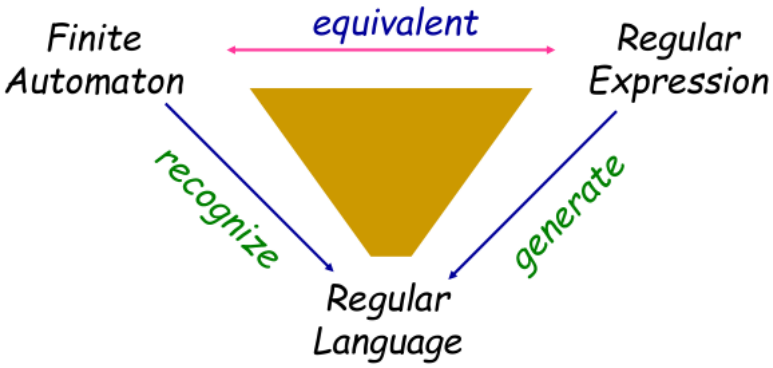
解：$1^*(011^*)^*(0+\varepsilon)$ 或 $(1+01)^*(0+\varepsilon)$

> **练习**：RegExp for ($\Sigma = \{0,1\}$)
>
> 1. {w | w has exactly a single 1 }
> 2. {w | w contains 001 }
> 3. {w | length(w) ≥ 3 and the third symbol is 0 }
> 4. What language does the RegExp $\phi^*$ represent ?

## 2. 正则表达式和FA的等价性

## 1. 等价性

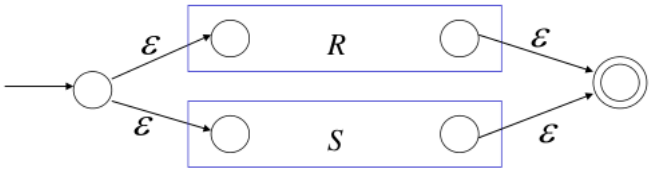
- 有穷自动机可以**识别**正则语言
- 正则表达式可以**生成**正则语言
- 故 有穷自动机和正则表达式**等价**



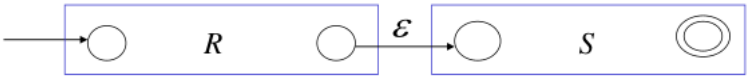

## 2. 正则表达式⇒FA

正则表达式三种运算的转换：通过空转移

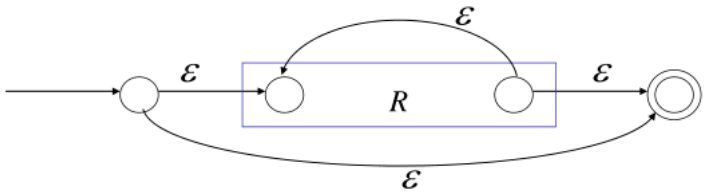
- 加：R+S



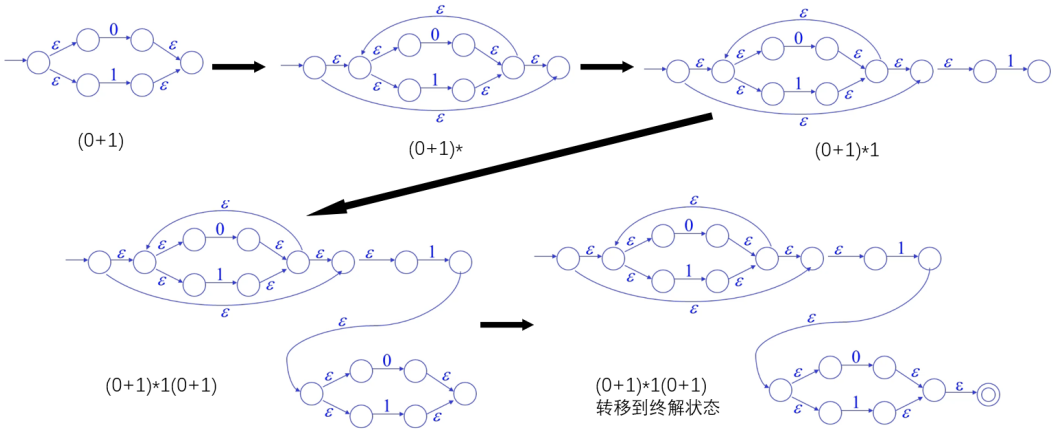

- 连接：RS




- 星：R*

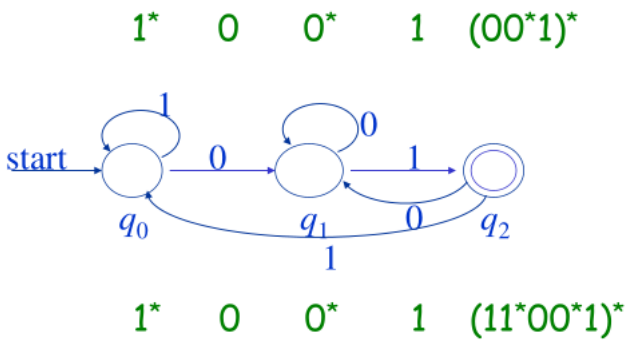



例：将 (0+1)*1(0+1) 转化为FA




## 3. FA⇒正则表达式

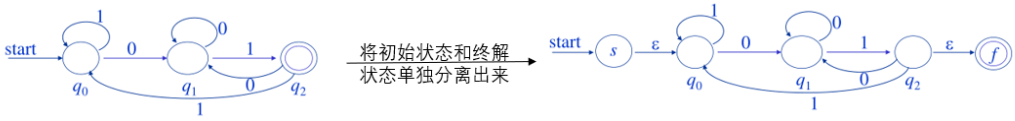
一些题目可以直接"看"出来，像这样：



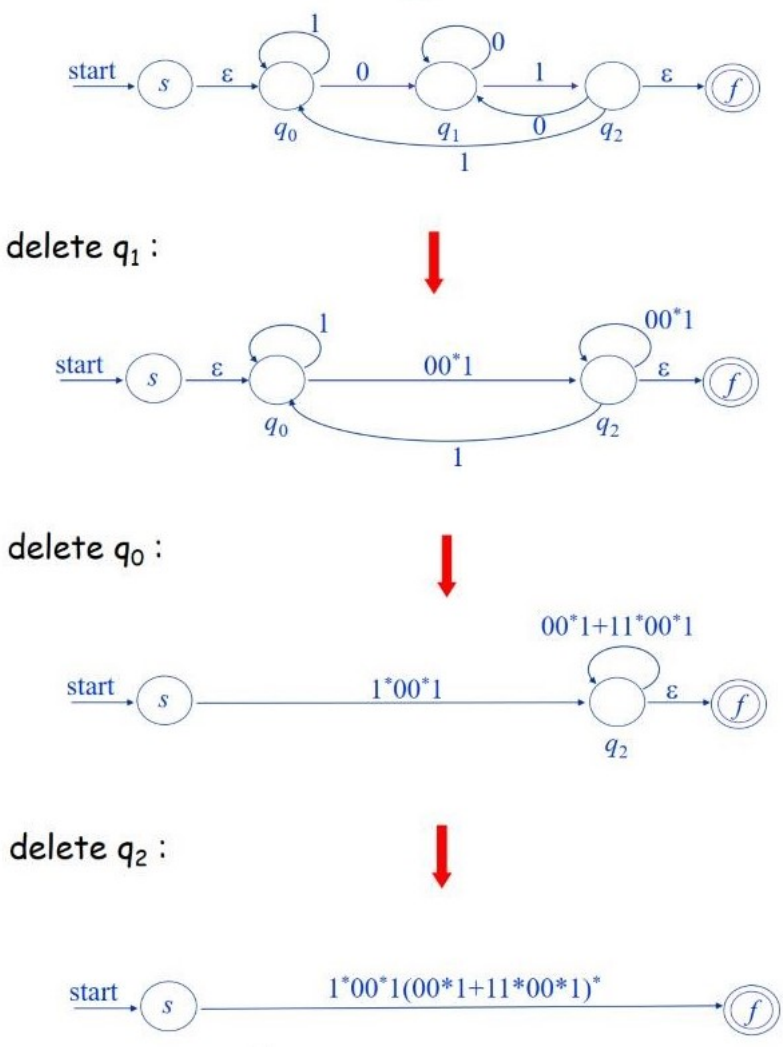

1*00*1 (00*1+11*00*1)*

### 状态消去法

但是题目太复杂了，容易丢三落四，于是使用**状态消去法。**过程如下：




然后每次消去一个状态，并将该状态表示的表达式添加在弧上




最后弧上得到的表达式就是该FA对应的正则表达式。

### 归纳法

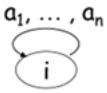
> 将状态标记为 Q={1, 2, 3, ....., n}
>
> $R_{ij}^{(k)}$ 其中$0 \le k \le n$：i 到 j 路径的正则表达式，且路径上没有标记超过 k 的状态

过程：按照k=0，i=j 的公式算出不经过中间状态的正则表达式，然后递推套k≥1的公式就行。

$k = 0$，i = j


$$\Rightarrow R_{ij}^{(0)} = \varepsilon + \phi$$

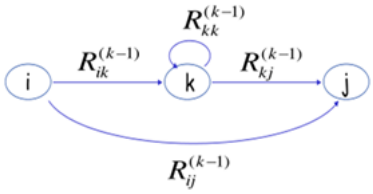

$$\Rightarrow R_{ij}^{(0)} = \varepsilon + a$$




$$\Rightarrow R_{ij}^{(0)} = \varepsilon + a_1 + a_2 + \cdots + a_n$$

$k \geq 1$

$$R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} (R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}$$




## 3. 正则语言的性质

### 1. 泵引理 (Pumping lemma)

> 泵引理用于证明某个语言**不是正则的**，它是正则语言应满足的必要条件
>
> 要证明**是正则的**可以使用构造DFA、NFA、ε-NFA、正则表达式的方法

**正则语言的泵引理**：假设 L 的是正则的，则 $\exists n$，对 $\forall w \in L$，若$|w| \geq n$，则可将 w 划分为 $w = xyz$，其中

- $|xy| \leq n$
- $y \neq \varepsilon (|y| \geq 1)$
- $\forall k \geq 0,\ xy^k z \in L$

例：证明 $L = \{0^n 1^n \mid n \geq 0\}$ 不是正则的

解：假设它是正则的。由泵引理可知，一定存在一个常数n，对每个长度不小于n的$w \in L$，可以被分成3的子串 $w = xyz$，且$|xy| \leq n$，$y \neq \varepsilon$，$xy^k z \in L$

取 $w = 0^n 1^n \in L$，则 $w = 0^n 1^n = xyz$，由于$|xy| \leq n$，故只能有 $y = 0^m$，所以有 $xz = 0^{n-|y|} 1^n \in L$。

显然xz不是L中的字符串，所以矛盾，所以L不是正则的。

> **解题方法**：上手先假设L是正则的，根据泵引理，选一个字符串w (最好是在开始的某个字符上重复了n次，即最开始是一个 $a^n$)，然后就可以泵掉 |y| 个 a，然后就不属于L了，就推出矛盾

### 2. 封闭性

正则语言经某些运算后得到的新语言**仍是正则语言**，称正则语言在这些运算下封闭。

- 并：$L \cup M$
- 交：$L \cap M$
- 补：$\overline{L}$
- 差：$L - M$
- 反：$L^R$
- 闭包(星)：$L^*$
- 连接：$LM$
- 同态：$h:\ \Sigma^* \to \Gamma^* \qquad h(L) = \{h(w) \mid w \in L\}$
- 逆同态

---

# CFG和PDA

## 1. 上下文无关文法 (CFG)

### 1. 形式化定义

**上下文无关文法 (Context-Free Grammars)**：CFG是一个四元组，如：$G = (V, T, S, P)$，其中

- $V$：变元的集合，是一个有限集；（变量）
- $T$：终结符的集合，是一个有限集，且 $V \cap T = \phi$；（值）
- $S$：开始变元，$S \in V$；
- $P$：产生式的集合，是一个有穷集，其中的每个元素都有形式：$A \to \alpha$，其中 $A \in V, \alpha \in (V \cup T)^*$

**派生**：由产生式生成字符串的过程。

- **最左派生**：每次选取派生式的最左的变元派生替换。
- **最右派生**：每次选取派生式的最右变元派生替换。

> 例如：$L = \{a^{2n}b^m | n \geq 0, m \geq 0\}$ 的产生式为：$S \to AB,\ A \to \varepsilon | aaA,\ B \to \varepsilon | Bb$
>
> 对于字符串 $w = aabb$ 来说，派生式如下：
>
> $S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb \Rightarrow aaBbb \Rightarrow aabb$
>
> - 最左派生：$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aaBbb \Rightarrow aabb$
> - 最右派生：$S \Rightarrow AB \Rightarrow ABb \Rightarrow ABbb \Rightarrow Abb \Rightarrow aaAbb \Rightarrow aabb$

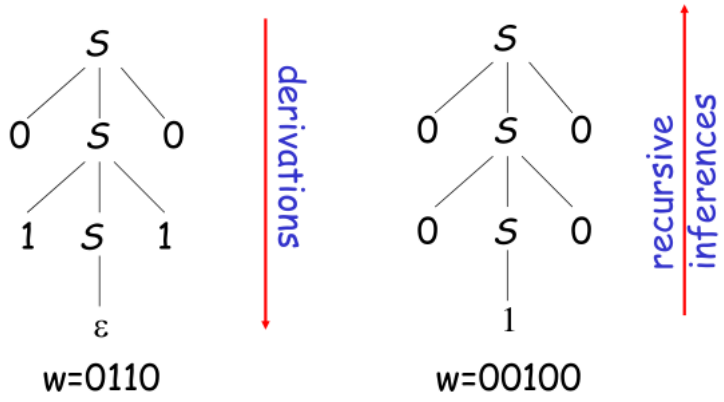
**上下文无关语言 (CFL)**：$G = (V, T, S, P)$ 是一个CFG，则 $L(G) = \{w \mid w \in T^* \ and \ S \overset{*}{\Longrightarrow} w\}$

## 2. 语法分析树

**语法分析树**：$G = (V, T, S, P)$ 是一个CFG，一个G的语法分析树如下：

- 每个内节点都标了一个 $V$ 中的变元；
- 每个叶节点都标了一个 $T \cup \{\varepsilon\}$ 中的符号，所有被 ε 标记的叶节点都是其父节点的唯一子节点；
- 如果一个内节点标记为A，它的子节点(从左到右)标记为 $x_1, x_2, \ldots, x_k$，则 $A \to x_1, x_2, \ldots, x_k \in P$

例：$L = \{w \mid w \in \{0,1\}^* \ and \ w = w^R\}$ 产生式为 $S \to \varepsilon \mid 0 \mid 1 \mid 0S0 \mid 1S1$ 两个语法分析树如下：



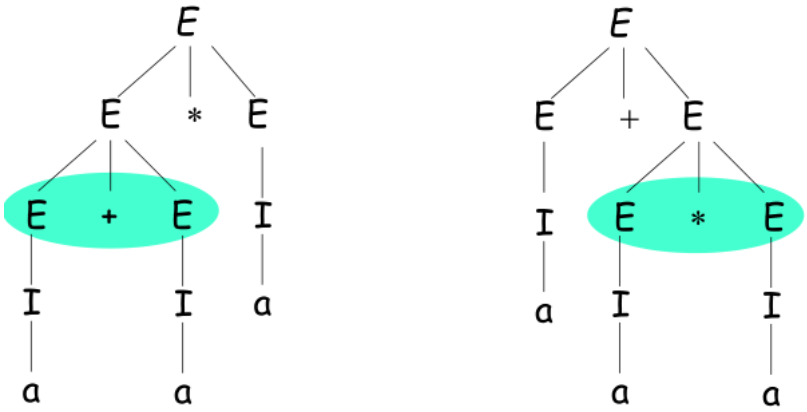

## 3. 二义性

对于一个CFG：$G = (\{E, I\}, \{a, b, (, ), +, *\}, E, P)$，产生式为 $E \to I \mid E + E \mid E * E \mid (E)$，$I \to a \mid b$

对于字符串 $w = a + a * a$ 的两种最左派生如下：

$E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow I + E * E \Rightarrow a + E * E \Rightarrow a + a * a$
$E \Rightarrow E + E \Rightarrow I + E \Rightarrow a + E \Rightarrow a + E * E \Rightarrow a + a * a$

对应的语法分析树如下，发现一个先算的是加法，一个先算的是乘法，出现了歧义。




重新构造产生式以消除歧义：

先算乘法的：$E \to I \mid E + E \mid E * E \mid (E),\ I \to a \mid b$

先算加法的：$E \to T \mid E + T,\ T \to F \mid T * F,\ F \to I \mid (E),\ I \to a \mid b \mid Ia \mid Ib$

> 定义同样的语言可以有多个文法，如果一个CFL的所有文法都是歧义的，那么它是固有二义性的

## 4. CFG的化简

- 去掉 ε 产生式；

- 去掉单元产生式；
- 去掉无用的产生式；

### 5. 乔姆斯基范式(CNF)

**乔姆斯基范式(Chomsky Normal Form)**：一个CFG的所有的产生式都有如下两种形式之一：

- $A \to BC$，$A, B, C \in V$
- $A \to a$，$a \in T$

CFG可以转换为CNF的形式，如下例子。

**例**：将 $S \to ABa, A \to aab, B \to Ac$ 转化为CNF的形式

**解**：$S \to AC, A \to DE, B \to AF, C \to BD, D \to a, F \to c, E \to DG, G \to b$

## 2. 下推自动机(PDA)

> 由于FA有局限性，可以识别$M = \{0^n 1^m | n \geq 0, m \geq 0\}$，但不能识别$L = \{0^n 1^n | n \geq 0\}$，所以有了PDA

### 1. 形式化定义

**下推自动机(Pushdown Automata)**：PDA是一个七元组$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$，其中，

- $Q$ 是有限的状态集；
- $\Sigma$ 是有限的输入字符集；
- $\Gamma$ 是有限的栈字符集；
- $\delta$ 是状态转移函数，是一个映射 $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \Rightarrow 2^Q \times \Gamma^*$；
- $q_0$ 是初始状态；
- $z_0$ 是初始栈符，表示栈是空的；
- $F$ 是终结状态集；

**例**：构造PDA识别 $L = \{ww^R | w \in \{0,1\}^*\}$

**解**：第一步，把 $w$ 入栈

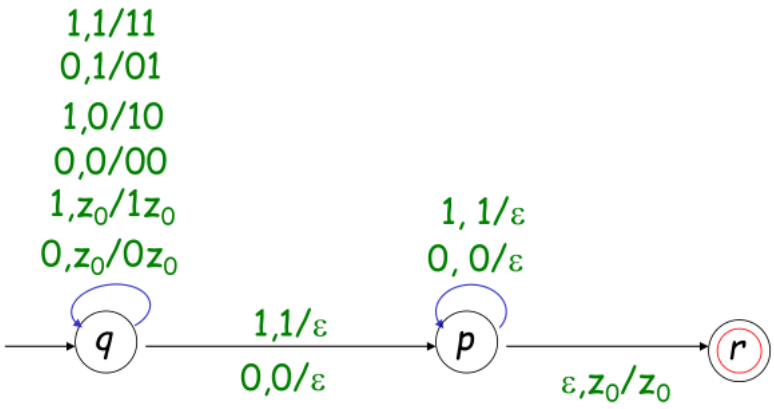
$$\begin{aligned} \delta(q,0,z_0) &= (q,0z_0), & \delta(q,1,z_0) &= (q,1z_0) \\ \delta(q,0,0) &= (q,00), & \delta(q,1,0) &= (q,10) \\ \delta(q,0,1) &= (q,01), & \delta(q,1,1) &= (q,11) \end{aligned}$$

第二步，从栈中弹出 $w^R$

$$\begin{aligned} \delta(q,1,1) &= (p,\varepsilon), & \delta(q,0,0) &= (q,\varepsilon) \\ \delta(p,1,1) &= (p,\varepsilon), & \delta(p,0,0) &= (q,\varepsilon) \end{aligned}$$

第三步，转移到终结状态 $\delta(p, \varepsilon, z_0) = (r, z_0)$

图示如下，*这是一个不确定的PDA*



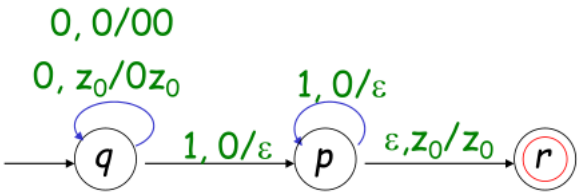

### 2. 确定的PDA

如果一个PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ 是确定的，那么它满足下面的条件：

- $\forall q \in Q, \forall a \in \Sigma \cup \{\varepsilon\}, \forall X \in \Gamma$，$\delta(q, a, X)$ 的结果是唯一的；
- $\delta(q, a, X)$ 和 $\delta(q, \varepsilon, X)$ 只能有一个有定义，因为对于状态$q$来说，读 $\varepsilon$ 意味着不读 $a$，而另一个意味着读 $a$，所以读与不读就产生了不确定性。

**例**：构造确定的PDA识别 $L = \{0^n 1^n | n > 0\}$

**解**：这就是一个DPDA




### 3. PDA的瞬时描述

用一个三元组 $(q, w, \alpha)$ 来描述一个PDA在某一时刻的格局，其中，

- $q$ 是PDA此时的状态；
- $w$ 是剩余的待读入字符串；
- $\alpha$ 是栈中的字符串。

**例**：用格局序列描述2中构造的 $L = \{0^n 1^n | n > 0\}$ 的PDA接受 $w = 0011$ 的过程。

**解**：$(q, 0011, z_0) \vdash (q, 011, 0z_0) \vdash (q, 11, 00z_0) \vdash (p, 1, 0z_0) \vdash (p, \varepsilon, z_0) \vdash (r, \varepsilon, z_0)$

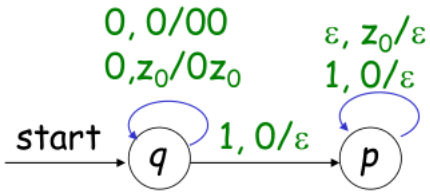
简记为 $(q, 0011, z_0) \vdash^* (r, \varepsilon, z_0)$

### 4. PDA接受的语言

能够由PDA接受(CFG构造)的语言称为**上下文无关语言(CFL)**，PDA可以用两种方式描述接受语言：

- 用**终结状态**来描述：$L(P) = \{w \,|\, (q_0, w, z_0) \vdash^* (q, \varepsilon, \alpha), q \in F\}$
- 用**空栈状态**来描述：$N(P) = \{w \,|\, (q_0, w, z_0) \vdash^* (q, \varepsilon, \alpha)\}$
- 这两种描述方式是**等价**的，即 $L(P) \Leftrightarrow N(p)$

例如2中构造的 $L = \{0^n 1^n | n > 0\}$ 的PDA就是用终结状态接受的，也可用空栈状态来描述，如下




但是并不是所有的PDA都可以用两种方式构造 (针对DPDA)，当 $L$ 可以**被终结状态的DPDA接受**并且 $L$ **有前缀性**的时候，$L$ 才能被空栈状态的DPDA接受。

> 语言的**前缀性**：该语言中没有两个不同的字符串x和y，使得x是y的前缀。
>
> 如：语言 0* 就没有前缀性，因为0是00的前缀。

## 3. CFG和PDA的等价性

对于一个给定的上下文无关语言 $L$，存在一个CFG生成 $L$，且存在一个PDA识别 $L$。

### 1. CFG $\Rightarrow$ PDA

把CFG $G = (V, T, S, P)$ 转化为PDA，则对应的PDA为 $B = (\{q\}, T, V \cup T, \delta, q, S, \{\})$，其中，

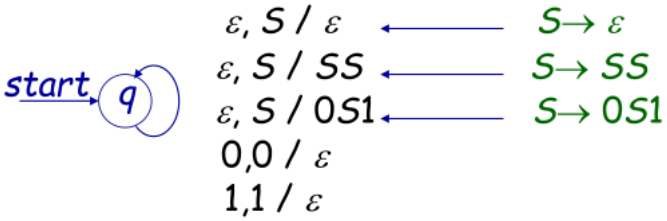
- $\delta(q, \varepsilon, A) = \{(q, \alpha) | A \to \alpha \in P\}$
- $\delta(q, a, a) = (q, \varepsilon)$

**例**：将CFG $G = (\{S\}, \{0, 1\}, \{S \to 0S1, S \to SS, S \to \varepsilon\}, S)$ 转化为PDA。

**解**：PDA为 $P = (\{q\}, \{0, 1\}, \{0, 1, S\}, \delta, q, S, \{\})$，其中 $\delta$ 定义如下：

- $\delta(q, \varepsilon, S) = \{(q, 0S1), (q, SS), (q, \varepsilon)\}$
- $\delta(q, 0, 0) = \{(q, \varepsilon)\}$
- $\delta(q, 1, 1) = \{(q, \varepsilon)\}$

用图表示




该PDA识别字符串 $w = 0011$ 的过程：

$(q, 0011, S) \vdash (q, 0011, 0S1) \vdash (q, 011, S1) \vdash (q, 011, 0S11) \vdash (q, 11, S11) \vdash (q, 11, 11) \vdash (q, 1, 1) \vdash (q, \varepsilon, \varepsilon)$

对应的CFG派生序列：$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0011$

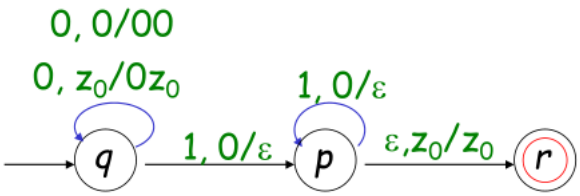
> 转化出的PDA实际上是在模拟CFG的派生过程，所以PDA一定能就识别CFG生成的字符串

## 2. PDA $\Rightarrow$ CFG

把PDA $P=(Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ z_0,\ F)$ 转化为CFG，则对应的CFG为 $G=(V,\Sigma,S,R)$，其中，

- $V$：包括开始变元 $S$，这个变元和PDA没有关系，就是强行规定的；还有其他形如 $[qXp]$ 的符号，其中$\forall q,p\in Q,X\in\Gamma$
  - 符号 $[qXp]$ 的意义是在 $q$ 状态下，可以**使栈中的 $X$ 弹出并转移到 $p$ 状态**的字符串，例如有状态转移函数 $\delta(q_0,\varepsilon,z_0)=(p,\varepsilon)$，则 $[q_0z_0p]\rightarrow\varepsilon$，于是对于下面 $R$ 的第一条产生式规则，就有$S\rightarrow[q_0z_0p]$
- $R$：包括 $\forall p\in Q$，$S\rightarrow[q_0z_0p]$；还有 $[qXr_k]\rightarrow a[rY_1r_1][r_1Y_2r_2]\dots[r_{k-1}Y_kr_k]$，对于 $(r,Y_1Y_2\dots Y_k)\in\delta(q,a,X)$
  - 第一条产生式规则已经在上一条中描述了，下面是关于第二条产生式规则。对于状态转移函数 $\delta(q,a,X)=(r,Y_1Y_2\dots Y_k)$，因为 $[qXr_k]$ 表示的是把 $X$ **全pop掉**所需要的字符串，而状态转移函数读入字符串 $a$ 之后栈中的元素是 $Y_1Y_2\dots Y_k$，所以需要把这些元素也pop掉，因此最后的状态就不是 $r$ 而是 $r_k$，而第二条产生式规则的body部分 $a$ 之后的部分就是做这个的。

**例**：还是用 *2.2确定的PDA* 中的例子，将其转化成CFG




**解**：$P=(Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ z_0,\ F)\Rightarrow G=(V,\Sigma,S,R)$，其中
$V=\{S,[qz_0q],[qz_0p],[q0q],[q0p],[q1q],[q1p],[pz_0q],[pz_0p],[p0q],[p0p],[p1q],[p1p]\}$

然后根据转移函数导出产生式 $R$

- $\delta(q,0,z_0)=(q,0z_0)\Rightarrow[qz_0r_2]\rightarrow0[q0r_1][r_1z_0r_2],\forall r_1,r_2\in Q\Rightarrow$
  $[qz_0q]\rightarrow0[q0q][qz_0q]\mid0[q0p][pz_0q]$
  $[qz_0p]\rightarrow0[q0q][qz_0p]\mid0[q0p][pz_0p]$
- $\delta(q,0,0)=(q,00)\Rightarrow[q0r_2]\rightarrow0[q0r_1][r_10r_2],\forall r_1,r_2\in Q\Rightarrow$
  $[q0q]\rightarrow0[q0q][q0q]\mid0[q0p][p0q]$
  $[q0p]\rightarrow0[q0q][q0p]\mid0[q0p][p0p]$
- $\delta(q,\varepsilon,z_0)=(p,z_0)\Rightarrow[qz_0r_1]\rightarrow[pz_0r_1],\forall r_1\in Q\Rightarrow$
  $[qz_0q]\rightarrow[pz_0q]$
  $[qz_0p]\rightarrow[pz_0p]$
- $\delta(q,1,0)=(p,\varepsilon)\Rightarrow[q0p]\rightarrow1$
- $\delta(p,1,0)=(p,\varepsilon)\Rightarrow[p0p]\rightarrow1$
- $\delta(p,\varepsilon,z_0)=(p,\varepsilon)\Rightarrow[pz_0p]\rightarrow\varepsilon$

把得到的产生式整合在一起得到 $R$

$$
\begin{gathered}
R=\{\quad S\rightarrow[qz_0q]\mid[qz_0p],\\
[qz_0q]\rightarrow0[q0q][qz_0q]\mid0[q0p][pz_0q],\\
[qz_0p]\rightarrow0[q0q][qz_0p]\mid0[q0p][pz_0p],\\
[q0q]\rightarrow0[q0q][q0q]\mid0[q0p][p0q],\\
[q0p]\rightarrow0[q0q][q0p]\mid0[q0p][p0p],\\
[qz_0q]\rightarrow[pz_0q].\quad[qz_0p]\rightarrow[pz_0p],\\
[q0p]\rightarrow1,\ [p0p]\rightarrow1,\ [pz_0p]\rightarrow\varepsilon\quad\}
\end{gathered}
$$

最后**把 $R$ 按如下规则化简**一下：

- 消除含有没有终结符的变元的产生式，如：含有 $[pz_0q]$ 的产生式；
- 消除死循环的产生式，如：$[q0q]$ 的第一个产生式，因为它的第二个产生式由于 $[p0q]$ 满足第一条化简规则，所以它只剩下第一个产生式，所以它死循环了；
- 消除含有由于前两条规则导致的无用变元的产生式，如：因为 $[q0q]$ 无用，所以含有它的产生式也无用。

最终得到

$$
\begin{gathered}
R=\{\quad S\rightarrow[qz_0p],\ [qz_0p]\rightarrow0[q0p][pz_0p],\\
[q0p]\rightarrow0[q0p][p0p],\ [qz_0p]\rightarrow[pz_0p],\\
[q0p]\rightarrow1,\ [p0p]\rightarrow1,\ [pz_0p]\rightarrow\varepsilon\quad\}
\end{gathered}
$$

看起来不太方便，于是令 $A=[qz_0p],B=[q0p],C=[p0p],D=[pz_0p]$，得到

$R=\{S\rightarrow A,\ A\rightarrow0BD|D,\ B\rightarrow1|0BC,\ C\rightarrow1,\ D\rightarrow\varepsilon\}$

再次化简得到：$R=\{S\rightarrow0B|\varepsilon,\ B\rightarrow1|0BC,\ C\rightarrow1\}$

# 4. 上下文无关语言的性质

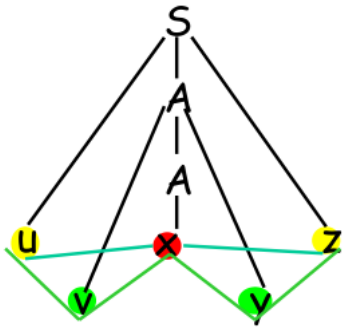
## 1. 泵引理

**上下文无关语言的泵引理**：$L$ 是一个CFL，则 $\exists n$，对 $\forall w\in L$，若 $|w|\geq n$，则 $w$ 可以划分为 $w=uvxyz$，其中

- $|vxy|\leq n$
- $|vy|\geq1$，(要是vy同时为空就出现 $A\rightarrow A$ 这种没有意义的产生式了)
- $uv^ixy^iz\in L,\ \forall i=0,1,2,\dots$

> n的取法：令 $m=|V|$，$k=max\{|\alpha|\forall A\to\alpha\}$，则 $n=k^m$

派生过程：$S\overset{*}{\Rightarrow}uAz\overset{*}{\Rightarrow}uvAyz\overset{*}{\Rightarrow}w$，语法解析树如下，对于重复出现的 A 来说，则可用子树的A代替父节点，此时失去的就是一对vy节点。




**例**：证明 $L=\{ww|w\in\{0,1\}^*\}$ 不是CFL。

**解**：假设L是CFL。则由泵引理可知，存在一个常数n，对于L中长度不小于n的字符串w就可以划分为五个部分，$w=uvxyz$，其中 $|vxy|\le n$，$vy\neq\varepsilon$，$uv^kxy^kz\in L$。

取 $w=0^n1^n0^n1^n\in L$，则 $uvxyz=0^n1^n0^n1^n$(如果要推出矛盾，就需要推出 $uxz\notin L$)。v和y不能同时为空串且 $|vxy|\le n$，所以它们的取值情况可以分为7种情况，这七种情况又可以分为两类：

- 第一类：vxy在同一类字符里，即同在开始的n个0、同时在开始的n个1里、同时在结束的n个0里，同时在结束的n个1里。这四种情况是等价的，而显然在第一种情况下有 $uxz\notin L$，因为开始的0的个数不足n了。
- 第二类：vxy在连续的两类字符里，即在前半部分的 $0^n1^n$ 中、在中间的 $1^n0^n$ 中、在后半部分的 $0^n1^n$中。这三种情况是等价的，而显然在第一种情况下有 $uxz\notin L$，因为开始的0和1的个数都不足n了。

所有的情况都推出了矛盾，所以假设错误，即 L 不是CFL。

## 2. 封闭性

CFL在**并、连接、星、反转、交、同态、逆同态**运算下是封闭的，而在**交、补**运算下不是封闭的。

> 对于两个CFL $L_1$ 和 $L_2$，令 $G(L_1)=(V_1,T_1,R_1,S_1),G(L_2)=(V_2,T_2,R_2,S_2)$
>
> - 并：$G(L_1\cup L_2)=(V_1\cup V_2\cup\{S\},T_1\cup T_2,R,S)$，$R=\{S\to S_1|S_2\}\cup R_1\cup R_2$
> - 连接：$G(L_1\cup L_2)=(V_1\cup V_2\cup\{S\},T_1\cup T_2,R,S)$，$R=\{S\to S_1S_2\}\cup R_1\cup R_2$
> - 星：$G(L_1^*)=(V_1,T_1,\{S_1\to S_1S_1|\varepsilon\}\cup R_1,S_1)$
> - 反转：$G(L_1^R)=(V_1,T_1,\{A\to\alpha^R|A\to\alpha R_1\},S_1)$

交运算不封闭，例如：$L_1=\{a^nb^nc^m|n\ge0,m\ge0\}$，$L_2=\{a^nb^mc^m|n\ge0,m\ge0\}$ 是两个CFL，它们的交就是 $L_1\cup L_2=\{a^nb^nc^n|n\ge0\}$，这不是CFL，可以按照上面的方式用泵引理证明。

> 但是一个CFL和一个RL做交运算之后得到的还是CFL，这个条件下它是封闭的。

---

# 图灵机 (Turing Machine)

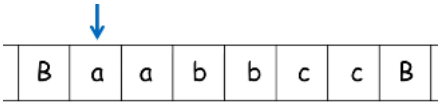
## 1. 形式化定义

**图灵机(Turing Machine)**：TM是一个七元组$P=(Q,\Sigma,\Gamma,\delta,q_0,B,F)$，其中，

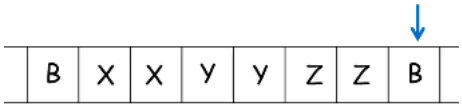
- $Q$ 是有限的状态集；
- $\Sigma$ 是有限的输入字符集；
- $\Gamma$ 是有限的纸带字符集；
- $\delta$ 是状态转移函数，是一个映射 $Q\times\Gamma\Rightarrow Q\times\Gamma\times\{R,L\}$，状态转移函数 $\delta(q,X)=(p,Y,D)$ 表示状态从q到p，读写头所指的字符X被改为Y，读写头向D方向移动；
- $q_0$ 是初始状态；
- $B$ 是空格符，一个特殊的符号；
- $F$ 是终结状态集；

**例**：构造TM识别 $L=\{a^nb^nc^n|n\ge0\}$

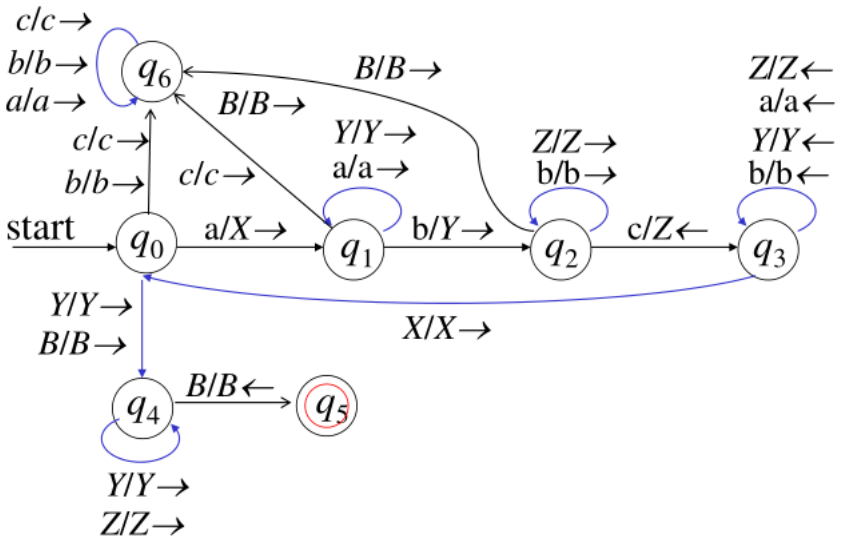
初始状态：




最终状态：




构造的图灵机：$M=(\{q_0,q_1,q_2,q_3,q_4\},\{a,b,c\},\{a,b,c,B,X,Y,Z\},\delta,q0,B,\{q_4\})$




> 上图表示的是确定的图灵机，删除 $q_6$ 得到的是不确定的图灵机，也可以识别语言 L

### 1. 瞬时描述

用 $X_1 \dots X_{i-1} q X_i X_{i+1} \dots X_n$ 的形式表示图灵机在某一时刻所处的格局，q表示图灵机所处的状态，读写头指向的位置为状态符q右侧的字符，即 $X_i$ 。

如上的图灵机识别字符串 $aabbcc$ 的过程描述如下：
$q_0 aabbcc \vdash X q_1 abbcc \vdash X a q_1 bbcc \vdash X a Y q_2 bcc X a Y b q_2 cc \vdash X a Y q_3 b Z c \vdash X a q_3 Y b Z c \vdash X q_3 a Y b Z c \vdash q_3 X a Y b Z c \vdash X q_0 a Y b Z c \vdash X X q_1 Y b Z c \vdash X X Y q_1 b Z c \vdash X X Y$

### 2. 停机

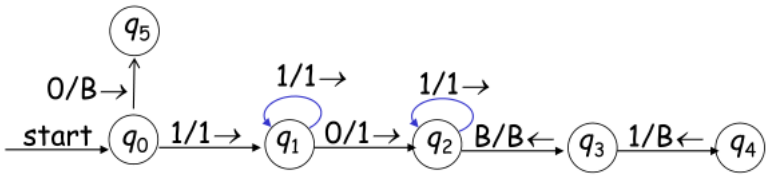
读了某个字符X之后，图灵机仍在状态q，不做任何动作，没有进行状态转移，简言之就是在状态q读入X没有相应的状态转移函数 。

- 不知道读写头向左/右移动；
- 不知道X应该被改成什么；
- 不知道转移到哪个状态；

## 2. 构造

**例**：设计一个TM计算两个正整数x和y的和，即 x+y。

**思路**：用x个1表示x，用y个1表示y，中间用0分隔。运算过程为，从左到右移动读写头，经过0则把它改成1，右移到B则左移一次，把1改成B，结束。状态转移图如下：




**例**：设计TM计算 $f(w) = ww,\ w \in \{1\}^+$ 。

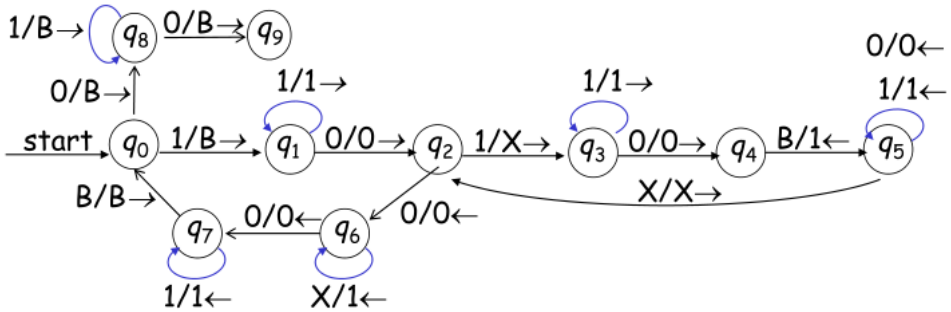
**思路**：用0分隔原来的w和复制得到的w，读一个1写一个1，并把读过的1改成X，复制结束后把X改回1，最后把左边的1右移一格，或者把右边的1左移一格即可（使用上题的方式）。

**例**：设计TM计算 $f(w) = ww,\ w \in \{0,1\}^+$ 。

**思路**：用字符A分隔原来的w和复制得到的w，用状态表示所读的是1还是0，读0则到q0状态，读1则到q1状态，然后根据所处的状态决定是写0还是写1。其余的参照上题的思路。

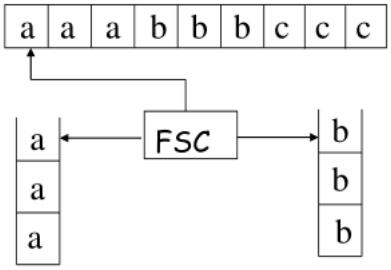
**例**：设计一个TM计算两个正整数m和n的积，即 m×n。

**思路**：做m个n相加，每加一个n，抹掉一个m的1，直到m的1全部被抹除。最后把与结果无关的全都抹除。



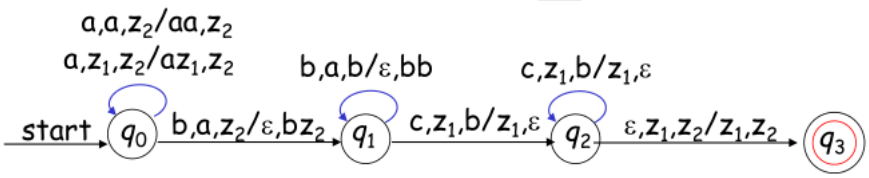
# 3. 双栈自动机 (Two Stack Machine)

状态转移函数：$\delta(q,a,X,Y)=(p,\alpha,\beta)$ 图示：




**例**：构造双栈自动机识别 $L=\{a^nb^nc^n|n\geq 0\}$

**解**：就是把所有的a先压栈，读b把b压栈的同时将a弹栈，最后用c把b弹栈




# 4. 图灵机编码

## 1. 字符串排序枚举

对 $w\in\{0,1\}^*$ 按照长度排序：$\varepsilon,0,1,00,01,10,11,000,001,010,011,\dots$

把上面的每个字符串前面用1连接得到：$1,10,11,100,101,110,111,1000,1001,1010,1011,\dots$

则可发现，若把 $1w$ 视为二进制数，则如果 $w$ 是第 $i$ 个字符串就有 $1w=i$

## 2. 编码

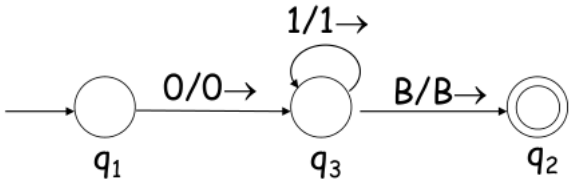
对于TM $M=(Q,\{0,1\},\Gamma,\delta,q_1,B,\{q_2\})$，其中 $Q=q_1,q_2,\dots,q_r$，$\Gamma=\{X_1,X_2,X_3,\dots,X_s\}$，并规定 $X_1=0,\ X_2=1,\ X_3=B,\ D_1=L,\ D_2=R$，$q_1$ 是初始状态，$q_2$ 是终结状态。

将转移函数编码为：$\delta(q_i,X_j)=(q_k,X_m,D_n)\Rightarrow 0^i10^j10^k10^m10^n$。**用 1 隔开**

则M可编码为：$C_111C_211C_311\dots C_{n-1}11C_n$，其中的 $C_i$ 表示状态转移函数的编码。**用 11 隔开**

> $C_i$ 之间是无序的，所以一个TM可以有不同的编码

**例**：编码如下TM




**解**：$\delta(q_1,0)=(q_3,0,\rightarrow)\Rightarrow 010100010100$
$\delta(q_3,1)=(q_3,1,\rightarrow)\Rightarrow 0001001000100100$
$\delta(q_3,B)=(q_2,B,\rightarrow)\Rightarrow 00010001001000100$

故 $TM\Rightarrow 010100010100\ 11\ 0001001000100100\ 11\ 00010001001000100$

**因为图灵机编码成了01字符串，而01字符串是可以枚举的，于是我们就可以称被编码的图灵机为第 i 个图灵机，记为 $M_i$ 。**

# 5. TM接受的语言

## 1. 递归可枚举语言

能够由图灵机接受的语言称为**递归可枚举语言(recursively enumerable language)**：$L=\{w\mid q_0w\vdash^*\alpha p\beta,\ p\in F,\ \alpha,\ \beta\in\Gamma^*\}$

> 所有的正则语言都是递归可枚举语言，所有的上下文无关语言都是递归可枚举语言。

## 2. 非递归可枚举语言

**定理**：$L_d=\{w_i\mid w_i\in L(M_i)\}$ 就是一个**非递归可枚举语言(not-recursively enumerable language)**，即没有TM接受它。

**证明**：假设 $L_d$ 是以一个TM $M$ 的接受的语言 $L(M)$，则我们可以假设 $M$ 的编码为 $w_i$，即 $M=M_i$。然后判断 $w_i$ 是否在 $L_d$ 里：

- $w_i\in L_d$：由假设可知，$M_i$ 接受 $L_d$，则一定接受其中的每个字符串，可得 $M_i$ 接受 $w_i$，由 $L_d$ 定义进而得 $w_i\notin L_d$；
- $w_i\notin L_d$：由假设可知，$M_i$ 接受的是 $L_d$，由于 $w_i\notin L_d$ 则 $M_i$ 不接受 $w_i$，由 $L_d$ 定义进而得 $w_i\in L_d$；

所以推出矛盾，假设错误。

## 3. 递归语言

如果存在TM $M$ 接受 $L$ 且满足下面两个条件，则称 $L$ 是**递归语言 (Recursive languages)**。

- $w \in L \Rightarrow M$ 接受 $w$ 并停机 (停在终结状态)；
- $w \notin L \Rightarrow M$ 停机 (停在非终结状态)；

**定理**：如果 $L$ 是递归语言，那么 $\overline{L}$ 也是。

**证明**：假设 $L = L(M), M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$，令 $\overline{M} = (Q \cup \{r\}, \Sigma, \Gamma, \delta, q_0, B, \{r\})$，其中

- r 是一个新的状态，不在Q中；
- 对于任意的 $q \in Q - F$ 和 $a \in \Sigma$，如果 $\delta(q, a) = \phi$，则有 $\delta(q, a) = (r, a, \rightarrow)$；

**定理**：如果 $L$ 和 $\overline{L}$ 都是递归可枚举语言，那么 $L$ 就是递归语言。

**证明**：假设 $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_1, B, F_1), M_2 = (Q_2, \Sigma, \Gamma, \delta_2, q_2, B, F_2)$ 分别接受 $L$ 和 $\overline{L}$，则令 $M = (Q_1 \times Q_2, \Sigma, \Gamma, \delta, (q_1, q_2), B, F_1 \times (Q_2 - F_2))$. $\delta((q, a), (a, b)) = (\delta_1(p, a), \delta_2(q, b))$ 即可用递归的方式接受 $L$，也就是读 $L$ 中的 $w$ 停在终结状态，读 $\overline{L}$ 中的 $w$ 停在非终结状态。

> 这个定理证明的时候构造的是一个有两个tape的图灵机。

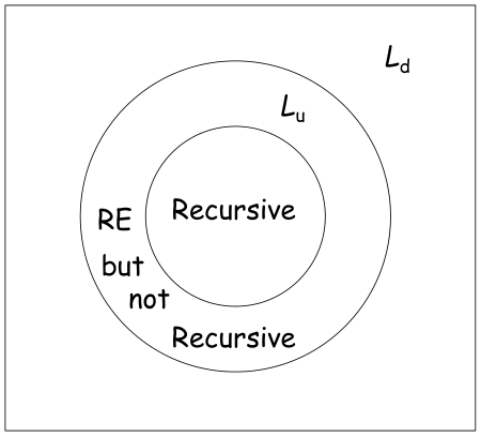
## 4. 通用语言

一个语言 $L$，它**是递归可枚举的但不是递归的**，即如果 $w \in L$ 则对应的 $M$ 接受它，但如果 $w \notin L$ 则 $M$ 不接受它且不会停机，则称它为**通用语言 (Universal language)**。$L_u = \{(M, w) | w \in L(M)\}$ 就是一个通用语言，其中 $M$ 是对应图灵机的编码，$w$ 是图灵机接受的字符串，连接成 $Mw$，记为 $(M, w)$。

**通用图灵机**：四条tape，以 $L(M) = \{0\}\{1\}^*$ 为例

- Tape1：图灵机的编码+111+接受的字符串；例：010100010100 **11** 0001001000100100 **11** 00010001001000100 **111** 011
- Tape2：接受的字符串编码；例：10100100，以1开头，用一个0表示0，两个0表示1
- Tape3：图灵机的状态；例：0，用0的个数表示状态
- Tape4：模拟图灵机的处理过程；

## 5. 语言的范畴




# 6. 乔姆斯基文法

乔姆斯基文法一共分为四型：

- Type 0：短语文法 phrase structure grammar(PSG) ------ 没有任何约束

  $\alpha \rightarrow \beta$;  $\alpha \in (V \cup T)^* V (V \cup T)^*, \beta \in (V \cup T)^*$
- Type 1：上下文有关文法 context sensitive grammar(CSG) ---- A左右为约定的字符串

  $\alpha A \beta \rightarrow \alpha \omega \beta$;  $A \in V,\ \alpha, \omega, \beta \in (V \cup T)^*$
- Type 2：上下文无关文法 context free grammar(CFG) ---- A左右都为空串

  $A \rightarrow \omega$;  $A \in V,\ \omega \in (V \cup T)^*$
- Type 3：正则文法 regular grammar(RG) ---- A只能生成**终结符**或有**一个变元且都在左(右)侧**的式子

  右线性文法 $A \rightarrow \alpha | \alpha B$;  $A, B \in V,\ \alpha \in T^*$ 或左线性文法 $A \rightarrow \alpha | B \alpha$;  $A, B \in V,\ \alpha \in T^*$

> 上下文有关文法对应上下文有关语言和线性有界自动机 (把图灵机中的空格符用 "[" 和 "]" 替换，处理的过程中读写头不能越过它们限定的范围)。