# Regular Expressions

## Regular expression

### Definition

Regular expressions describe language(regular languages)

If $E$ is a regular expression, $L(E)$ is the language it defines

RE 的定义是递归的。RE 使用三种语言上的基本操作：Union, Concatenation, Kleene Star

- Union: $L \cup M$
- Concatenation: $LM = \{wx : w \in L \text{ and } x \in M\}$
- Kleene Star: $L^* = \{\epsilon\} \cup L \cup LL \cup \ldots$

根据以上操作，即可给出 RE 的递归定义，RE 是 expression，由操作符和操作符连接起来的表达式构成，在给出 RE 的同时也将给出 RE 定义的语言

Basis.

- Basis 1: If $a$ is any symbol, then $a$ is a RE, $L(a) = \{a\}$
- Basis 2: $\epsilon$ is a RE, $L(\epsilon) = \{\epsilon\}$
- Basis 3: $\varnothing$ is a RE, $L(\varnothing) = \varnothing$

Induction.

- Induction 1: If $E_1, E_2$ are RE, $E_1 + E_2$ is a RE, $L(E_1 + E_2) = L(E_1) \cup L(E_2)$
- Induction 2: If $E_1, E_2$ are RE, $E_1 E_2$ is a RE, $L(E_1 E_2) = L(E_1)L(E_2)$
- Induction 3: If $E$ is a RE, then $E^*$ is a RE, $L(E^*) = (L(E))^*$
- Induction 4: if $E$ is a RE, then $(E)$ is a RE, $L((E)) = L(E)$

操作优先级由高到低是 $*, \text{concatenation}, +$

## Algebraic Laws for RE

- Union is commutative and associative, concatenation is associative
- Concatenation distributes over union
- $\varnothing$ is the identity for +, $\varnothing + R = R + \varnothing = R$

- $\epsilon$ is the identity for concatenation, $\epsilon R = R\epsilon = R$
- $\varnothing$ is the annihilator for concatenation, $\varnothing R = R\varnothing = \varnothing$
- Union is idempotent: $L + L = L$

## Laws Involving Closures

- $(L^*)^* = L^*$
- $\varnothing^* = \{\epsilon\}$
- $\{\epsilon\}^* = \{\epsilon\}$
- $(L + M)^* = (L^* M^*)^*$

如何判断一个 RE 的代数定律是否为真：代入一个具体的 RE （见书 3.4.7 节）
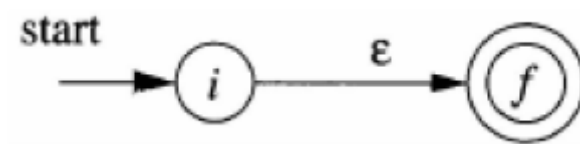
# Equivalence of RE and FA

## RE to FA

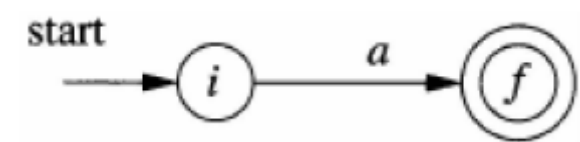For every RE, there is a FA that accepts the same language

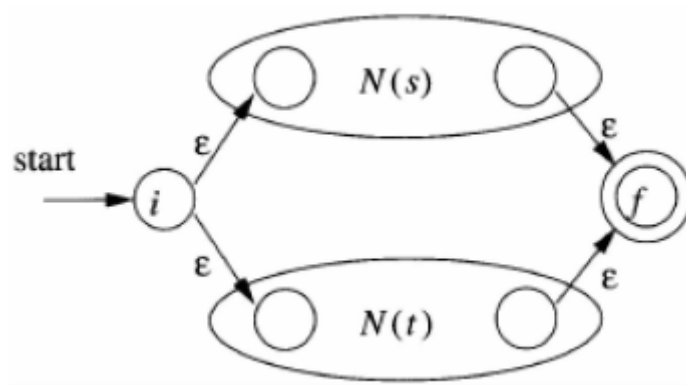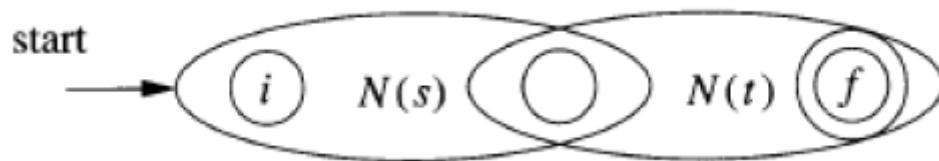Pick $\epsilon$-NFA

Basis.

$\epsilon$



$a$



Induction. 假设正则表达式 $s, t$ 的 NFA 为 $N(s), N(t)$

$s + t$

$st$



$s^*$

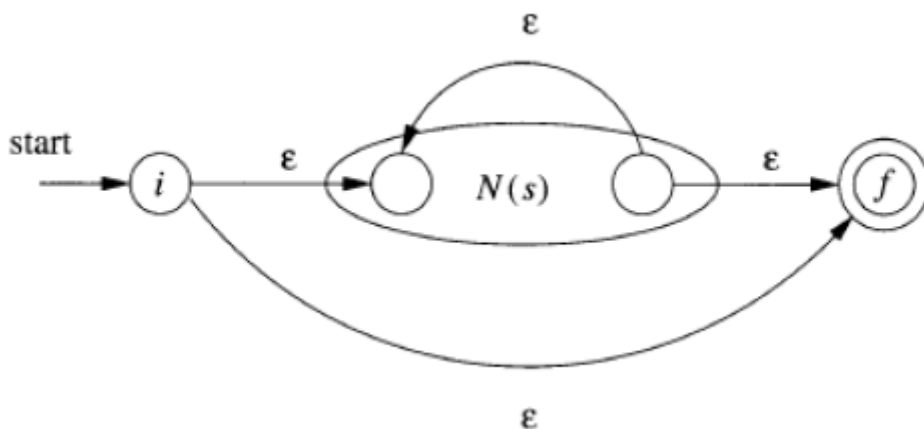

# FA to RE

For every FA, there is a RE defining its language

Pick DFA

使用归纳。假设 DFA 的状态为 $1, 2, \ldots, n$

定义 $k$-Path: A $k$-Path is a path through the graph of the DFA that goes through **no state numbered higher than $k$**

$k$-path 的 endpoint 没有限制，$n$-path 可以是任意路径

则该 DFA 对应的 RE 是所有从开始状态到接受状态的 $n$-path 的 RE 的集合

其正确性的证明基于对 $k$ 的归纳，令 $R_{ij}^k$ 为从状态 $i$ 到状态 $j$ 的 $k$-path 上的 label 表示的 RE

Basis. $k = 0$ ，此时 $R_{ij}^0$ 是从 $i$ 到 $j$ 所有边上的标号的和

- 若没有边，则为 $\varnothing$
- 若 $i = j$ ，则需要加上 $\epsilon$

Induction. 从 $i$ 到 $j$ 的 $k$-path，或者经过状态 $k$ （一次或多次），或者不经过状态 $k$

- 经过状态 $k$ ： $R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1}$
- 不经过状态 $k$ ： $R_{ij}^{k-1}$

故

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

The RE with the same language as the DFA is the union of $R_{ij}^n$ where

- $n$ is the number of states
- $i$ is the start state
- $j$ is one of the final states

# Properties of Regular Languages

## Properties

A **language class** is a set of languages

Language classes have two important kinds of properties

- Decision properties （判定性）
- Closure properties （封闭性）

Closure properties: given languages in the class, an operation produces another language in the same class

**Decision properties**: an **algorithm** that takes a formal description of a language and tells whether or not some property holds

- Membership problem: is string $w$ in regular language $L$
- Emptiness problem: does the language contain any string at all

## Pumping Lemma

Let $L$ be a regular language. Then there exists a constant $n$ (which depends on $L$) such that for every string $w$ in $L$ such that $|w| \geqslant n$ , we can break $w$ into three strings, $w = xyz$ , such that:

1. $y \neq \epsilon$
2. $|xy| \leqslant n$
3. For all $k \geqslant 0$, the string $xy^k z$ is also in $L$

pumping lemma 可以用于证明一个 language 不是 RL

# Decision Properties of Regular Language

## The membership problem

Problem: Given a string $w$ and a regular language $L$ , is $w$ in $L$

Algorithm: Simulate the DFA, if the DFA ends in an accepting state, the answer is "yes", otherwise the answer is "no"

## The emptiness problem

Problem: Given a regular language, does the language contain any string at all

Algorithm: Compute the set of states **reachable** from the start state. If at least one final state is reachable, then yes, else no.

> Basis. The start state is surely reachable from the start state.
>
> Induction. If state $q$ is reachable from the start state, and there is an arc from $q$ to $p$ with any label (input or $\epsilon$), then $p$ is reachable

如果语言由 RE 给出，则可根据以下递归规则判断是否为空

> Basis. $\varnothing$ denotes the empty language, $\epsilon$ and $a$ for any input symbol $a$ do not.
>
> Induction. Suppose $R$ is a RE. There are four cases to consider.
>
> 1. $R = R_1 + R_2$. Then $L(R)$ is empty $\iff$ $L(R_1)$ is empty **and** $L(R_2)$ is empty
> 2. $R = R_1 R_2$. Then $L(R)$ is empty $\iff$ $L(R_1)$ is empty **or** $L(R_2)$ is empty
> 3. $R = R_1^*$. Then $L(R)$ is not empty, it always includes at least $\epsilon$
> 4. $R = (R_1)$. Then $L(R)$ is empty $\iff$ $L(R_1)$ is empty

## The infiniteness problem

Problem: Is a given language infinite?

Key Idea: if the DFA has $n$ states, and the language contains ant string of length $n$ or more, then the language is infinite.

> Suppose that $L$ is a regular language accepted by a DFA with $n$ states. Then $L$ is infinite $\iff$ $L$ contains a word whose length is at least $n$.
>
> $\Leftarrow$: 如果一个有 $n$ 个状态的 DFA 接受一个长度至少为 $n$ 的字符串 $w$，则在其路径上至少有一个状态出现两次（PHP）。设其路径为
>
> $$s_1 s_2 \ldots s_i \ldots s_i \ldots s_n$$
>
> 设从 $s_1$ 到第一个 $s_i$ 的标号组成串 $x$，从第一个 $s_i$ 到第二个 $s_i$ 的标号组成串 $y$，从第二个 $s_i$ 到 $s_n$ 组成字符串 $z$，则该 DFA 可接受所有形如 $xy^i z (i \geqslant 0)$ 的字符串 (i. e. pumping lemma)
>
> $\Rightarrow$: trivial

However, there are an infinite number of strings of length $> n$, and we can't test them all

Key Idea: if there is a string of length $\geqslant n$ in $L$, then there is a string of length between $n$ and $2n - 1$

> Suppose that $L$ is a regular language accepted by a DFA with $n$ states. Then $L$ is infinite $\iff$ $L$ contains a word whose length is between $n$ and $2n - 1$.
>
> $\Rightarrow$: 因为 $L$ 是 infinite，其包含一个 string 长度至少为 $n$，设 $w$ 为 $L$ 包含的长度至少为 $n$ 的 string 中最短的
>
> $$w = w_1 w_2 \ldots w_N$$
>
> 设 DFA 接受其的状态顺序为 $a_0 a_1 \ldots a_N$，根据 PHP，其中必有一个状态至少出现两次，设其为 $a_i$，且 $a_i = a_j (i < j)$，故 $z = w_1 w_2 \ldots w_i w_{j+1} \ldots w_N$ 也属于 $L$。根据 $w$ 的定义，有 $|z| < n$，又 $|w| = z + j - i$ 且 $j - i \leqslant n$，故 $|w| < 2n$。根据其定义 $|w| \geqslant n$，故可得
>
> $$n \leqslant |w| \leqslant 2n - 1$$
>
> $\Leftarrow$: 同上（pumping lemma）

根据以上两个 key idea，检查所有长度在 $n$ 到 $2n - 1$ 之间的串即可

然而在实际中这样的算法代价也是不可接受的。所以一般是通过检查 DFA 对应的图中是否有环来判断语言是否 infinite。具体而言分为以下几步

1. Eliminate states not reachable from the start state
2. Eliminate states that do not reach a final state
3. Test if the remaining transition graph has any cycles

## The equivalence problem

Problem: Given regular languages $L$ and $M$, is $L = M$

Algorithm: Constructing the **product DFA** from DFA for $L, M$

设 $L$ 与 $M$ 的 DFA 有状态集合 $Q$ 与 $R$，则 product DFA 有状态集合 $Q \times R$

对于 product DFA 而言

- start state: $[q_0, r_0]$
- transition: $\delta([q, r], a) = [\delta_L(q, a), \delta_M(r, a)]$
- final states: states $[q, r]$ that exactly one of $q$ and $r$ is a final state of its own DFA

由于 product DFA 的 accept state 是两个状态中有且仅有一个是原本的 accept state，则如果一个 string $w$ 被 product DFA 接受，说明其被一个 DFA 接受而不被另一个 DFA 接受

$L = M \iff$ the product DFA's language is empty

## The containment problem

Problem: Given regular languages $L$ and $M$, is $L \subseteq M$

Algorithm: use the product DFA

final state: states $[q, r]$ that $q$ is the final state, $r$ is not

$L \subseteq M \iff$ the product DFA's language is empty

# The Minimum-State DFA for a Regular Language

## Equivalence of states

Given a DFA $A$, find the DFA with the fewest states accepting $L(A)$ (equivalence)

> Equivalence of states: states $p$ and $q$ are *equivalent* if
>
> For all input strings $w$, $\delta(p, w)$ is an accepting state $\iff \delta(q, w)$ is an accepting state
>
> If two states are not equivalent, then we say they are *distinguishable*. That is, state $p$ is distinguishable from state $q$ if there is at least one string $w$ such that one of $\delta(p, w)$ and $\delta(q, w)$ is accepting, and the other is not accepting.

Algorithm: table-filling algorithm

- Construct a table with all pairs of states

- If you find a string that **distinguishes** two states (takes exactly one to an accepting state), mark that pair
- Algorithm is a **recursion** on the length of the shortest distinguishing string

> Basis. If $p$ is an accepting state and $q$ is non-accepting, then the pair $[p, q]$ is distinguishable
>
> Induction. Let $p$ and $q$ be states such that for some input symbol $a$, $r = \delta(p, a)$ and $s = \delta(q, a)$ are a pair of states known to be distinguishable. Then $[p, q]$ is a pair of distinguishable states.

在算法结束后，所有未被标记的 pair 是等价的状态

> If two states are not distinguished by the table-filling algorithm, then the states are equivalent.
>
> Proof. 假设命题错误，故存在至少一对状态 $p, q$ 满足
>
> - $p, q$ is distinguishable.
> - the algorithm does not find $p$ and $q$ to be distinguishable
>
> 令这样的状态对为 bad pair。则对于所有 bad pairs，都存在一个 string 来区分两个状态。令其中 string 最短的 pair 为 $[p, q]$，区分其的最短 string 为 $w = a_1 a_2 \ldots a_n$
>
> 显然 $w \neq \epsilon$，否则在 Basis 阶段就被区分，即 $n \geqslant 1$。考虑 $r = \delta(p, a_1), s = \delta(q, a_1)$，显然，$a_2 a_3 \ldots a_n$ 区分 $[r, s]$，但是 $a_2 a_3 \ldots a_n$ 比任何能区分 bad pair 的 string 都短，故 $[r, s]$ 不是 bad pair，算法将其标记为 distinguishable。于是根据 Induction 的部分，算法发现 $[p, q]$ 是 distinguishable，因为 $r = \delta(p, a_1), s = \delta(q, a_1)$ 是 distinguishable。矛盾。故原命题正确

> 对于 Equivalence problem，也可以通过这个算法解决。若判断两个 regular languages $L, M$ 是否等价，可以将其状态合并，即 $Q \cup R$，然后对其应用 table-filling algorithm
>
> $L = M \iff$ 两个 DFA 的开始状态 $q_0, r_0$ 等价

## Minimize DFA

要最小化一个 DFA，首先要合并所有不可区分的状态

状态的合并可按照如下步骤

- 假设 $q_1, q_2, \ldots q_k$ 是不可区分的
- 将其用一个代表状态 $q$ 代替
- 显然 $\delta(q_1, a), \delta(q_2, a), \ldots \delta(q_k, a)$ 也是不可区分的（否则其中必有至少一对被算法标记为 distinguishable）

- 令 $\delta(q,a)$ 为 $\{p : p = \delta(q_i, a), i = 1, 2, \ldots k\}$ 的代表元素

合并之后，还要去掉所有从开始状态不可达的状态

事实上，状态的等价是一种等价关系，所以所有等价的状态将状态集划分为多个不相交的等价类

> The equivalence of states is transitive
>
> Proof. 假设 $[p,q]$ 与 $[q.r]$ 是等价的，但 $[p,r]$ 不等价。根据定义，存在一个输入 string $w$ 满足 $\delta(p,w)$ 与 $\delta(r,w)$ 中有且仅有一个是接收状态
>
> w. l. o. g. $\delta(p,w)$ 是接收状态。考虑 $\delta(q,w)$，若其为接收状态，$[q,r]$ 为 distinguishable，若其为非接受状态，$[p,q]$ 为 distinguishable。均与假设矛盾

故最小化 DFA 即为

- 消去所有从开始不可达的状态
- 将状态根据等价关系（由 table-filling algorithm 得到）划分为等价类
- 将含有不止一个状态的等价类合并为一个代表状态
- 合并状态转移

包含开始状态的等价类为新 DFA 的开始状态，包含接受状态的等价类为新 DFA 的接收状态

## Minimized DFA can't be beaten

> If $A$ is a DFA, and $M$ the DFA constructed from $A$ by the algorithm of minimizing DFA, then $M$ has as few states as any DFA equivalent to $A$

事实上，对于一个 DFA $A$，其所有最小的 DFA 都是同构的

> Proof. 假设 $A$ 是一个 DFA，最小化其得到 DFA $M$，假设存在一个 DFA $N$ 与 $A, M$ 接受一样的 language，但有更少的状态。考虑 $M, N$ 组合的 DFA，首先
>
> - $M, N$ 的开始状态是等价的，因为 $L(M) = L(N)$
> - 若状态 $[p,q]$ 等价，则 $[\delta(p,a), \delta(q,a)]$ 等价
>
> > 对于每个 $M$ 中的状态 $q$，其与 $N$ 中至少一个状态等价:
> >
> > Proof: by induction
> >
> > Basis. 如果 $q$ 是 $M$ 的开始状态，其与 $N$ 的开始状态等价
> >
> > 假设从 $M$ 的开始状态到达 $q$ 的最短路径长度为 $k$
> >
> > I. H. 若从 $M$ 的开始状态到达 $q$ 的最短路径长度短于 $k$，则存在一个 $N$ 中的状态与其等价

> Induction. 假设从 $M$ 的开始状态到 $q$ 的最短路径的 string 为 $w = xa$，长为 $k$。假设从 $M$ 开始状态出发沿 $x$ 到达 $r$，从 $N$ 开始状态出发沿 $x$ 到达 $p$，根据 I. H.，$[p, r]$ 等价。故 $[\delta_M(r, a), \delta_N(p, a)]$ 等价

由于根据前提，$N$ 状态数少于 $M$，故至少有两个 $M$ 中的状态与 $N$ 中的同一个状态等价。因此这两个状态等价，这与 $M$ 中所有状态都不等价矛盾，故前提错误，不存在这样的 $N$

# Closure Properties of Regular Language

## Closure Under Union

If $L$ and $M$ are regular languages, then so is $L \cup M$

> Proof. 如果 $L, M$ 为正则语言，则其有对应的 RE $R, S$，即
> $L = L(R), M = L(S)$，根据 RE 的定义，有
> $L \cup M = L(R) \cup L(S) = L(R + S)$，显然 $L \cup M$ 为正则语言

Concatenation 与 Kleene Closure 的证明同理，根据 RE 的定义即可证明其封闭性：$RS$ 是 $LM$ 的 RE，$R^*$ 是 $L^*$ 的 RE

## Closure Under Complementation

If $L$ is a regular language over alphabet $\Sigma$, then $\overline{L} = \Sigma^* - L$ is also a regular language

> Proof. Let $L = L(A)$ for some DFA $A = (Q, \Sigma, \delta, q_0, F)$, then $\overline{L} = L(B)$, where $B$ is the DFA $(Q, \Sigma, \delta, q_0, Q - F)$. 即构造一个 DFA $B$ 使得其所有接收状态都为 $A$ 中的非接受状态，这样对于任意字符串 $w$ 都有
>
> $$w \in L(B) \iff \delta(q_0, w) \in Q - F \iff \delta(q_0, w) \notin F \iff w \notin L(A)$$
> $$L(B) = \Sigma^* - L = \overline{L}$$

即 $\overline{L}$ 是由 DFA $B$ 定义的正则语言

## Closure Under Insertion

If $L$ and $M$ are regular languages, then so is $L \cap M$

> Proof. 可以通过 De Morgan's Law 以及 Complementation 和 Union 的封闭性证明
>
> $$L \cap M = \overline{\overline{L} \cup \overline{M}}$$

也可通过直接构造 DFA 证明。设 $L, M$ 对应的 DFA 为 $A, B$，则构造 product DFA $C$，其中

$$[p, q] \in F_C \iff p \in F_A \text{ and } q \in F_B$$

这样对于任意字符串 $w$ 有

$$\begin{aligned} w \in L(C) &\iff \delta([p_0, q_0], w) = [p, q] \\ &\iff \delta(p_0, w) = p \text{ and } \delta(q_0, w) = q \\ &\iff w \in L(A) \text{ and } w \in L(B) \\ &\iff w \in L(A) \cap L(B) \end{aligned}$$

则有 $L \cap M = L(C)$

已知 $L_1 = \{0^n 1^n : n \geqslant 0\}$ 不是 RE

$L_2$ 为有相等个 0 和 1 的串的集合，则 $L_2$ 不是 RE

Proof. 若 $L_2$ 是 RE，则 $L_2 \cap L(0^*1^*) = L_1$ 为 RE，矛盾，故 $L_2$ 不是 RE

## Closure Under Difference

If $L$ and $M$ are regular languages, then so is $L - M$

Proof. $L - M = L \cap \overline{M}$ ，根据 complementation 与 insertion 的封闭性，$L - M$ 也是 RE

## Closure Under Reversal

对于一个字符串 $w = a_1 a_2 \dots a_n$ ，其 reversal $w^R = a_n a_{n-1} \dots a_1$

对于一个语言 $L$ ，$L^R = \{w^R : w \in L\}$

If $L$ is a regular language, so is $L^R$

Automaton-based proof

对于 $L$ 的 DFA $A$

1. 将 $A$ 的转换图中的边反转
2. 令 $A$ 的开始状态为唯一的接收状态
3. 添加一个开始状态 $q_0$ ，以及从 $q_0$ 到其他接收状态的 $\epsilon$-transition

则 $w^R \in L(A^R) \iff w \in L(A)$

RE-based proof

假设 $L$ 的 RE 为 $E$，存在一个 RE $E^R$ 使得 $L(E^R) = (L(E))^R$。对 $E$ 的长度归纳

Basis. $\{\epsilon\}^R = \{\epsilon\}, \{a\}^R = \{a\}$

Induction.

- $E = E_1 + E_2$ , then $E^R = E_1^R + E_2^R$
- $E = E_1 E_2$ , then $E^R = E_2^R E_1^R$
- $E = E_1^*$ , then $E^R = (E_1^R)^*$

## Closure Under Homomorphisms

A string homomorphism is a **function** on strings that works by substituting a particular string for each symbol

If $h$ is a homomorphism on alphabet $\Sigma$, and $w = a_1 a_2 \ldots a_n$ is a string of symbols in $\Sigma$ ,then $h(w) = h(a_1)h(a_2)\ldots h(a_n)$

For language $L$

$$h(L) = \{h(w) : w \in L\}$$

If $L$ is a regular language over alphabet $\Sigma$ , and $h$ is a homomorphism on $\Sigma$, then $h(L)$ is also regular.

> Proof. 令 $R$ 为 $L$ 的 RE，对 $R$ 应用 $h$ 得到 $h(R)$，则 $h(R)$ 定义语言 $h(L)$ ，证明为对 $R$ 的子表达式 $E$ 归纳证明 $h(L(E)) = L(h(E))$
>
> Basis.
>
> 如果 $E$ 是 $\epsilon$ 或 $\varnothing$ ，则 $h(E) = E$ ，因为 $h$ 不影响 $\epsilon$ 或 $\varnothing$ 。显然，$L(h(E)) = L(E)$ 。又因为 $L(E)$ 为 $\{\epsilon\}$ 或 $\{\}$ ，则 $h(L(E)) = L(E)$ ，故 $L(h(E)) = h(L(E))$
>
> 如果 $E$ 是 $a$，$L(E) = \{a\}$ ，根据定义，$h(L(E)) = \{h(a)\}$ ，同理，$h(E)$ 是符号 $h(a)$ 组成的 string，则 $L(h(E)) = \{h(a)\}$ ，故 $L(h(E)) = h(L(E))$
>
> Induction.
>
> Union: 根据定义，有 $E = F + G, h(E) = h(F + G) = h(F) + h(G)$ ，根据 + 的定义
>
> $$L(E) = L(F) \cup L(G)$$
> $$L(h(E)) = L(h(F) + h(G)) = L(h(F)) \cup L(h(G))$$
>
> 又因为 $h$ 独立地应用于语言中的每个字符串，有
>
> $$h(L(E)) = h(L(F) \cup L(G)) = h(L(F)) \cup h(L(G))$$
>
> 根据 I. H.，$L(h(F)) = h(L(F)), L(h(G)) = h(L(G))$ ，故
>
> $$h(L(E)) = L(h(E))$$
>
> Concatenation 与 Kleene star 的证明类似

## Closure Under Inverse Homomorphisms

Suppose $h$ is a homomorphism from alphabet $\Sigma$ to strings in another (possibly the same) alphabet $T$. Let $L$ be a language over alphabet $T$. Then

$$h^{-1}(L) = \{w : w \in \Sigma^*, h(w) \in L\}$$

If $h$ is a homomorphism from alphabet $\Sigma$ to alphabet $T$, and $L$ is a regular language of $T$, then $h^{-1}(L)$ is also a regular language

Proof. 令 $L = L(A)$，其中 $A = (Q, T, \delta, q_0, F)$。定义 DFA

$$B = (Q, \Sigma, \gamma, q_0, F)$$

其中转换函数定义为 $\gamma(q, a) = \delta(q, h(a))$

$B$ 对于输入 $a$ 的转换的结果是 $A$ 对于输入串 $h(a)$ 的转换结果。根据字符串 $w$ 的长度归纳，可以轻松地得出 $\gamma(q_0, w) = \delta(q_0, h(w))$，由于 $A, B$ 的接收状态相同，故

$$w \in L(B) \iff h(w) \in L(A)$$
$$L(B) = \{w : w \in \Sigma^*, h(w) \in L\} = h^{-1}(L)$$