Chapter 1

automata abstract

1.1 Finite automata

Definition 1.1 (Finite automation). A finite automaton(an FA) is a 6-tuple (Q, V, T, E, S, F) where

- Q is a finite set of states,
- V is an alphabet,
- $T \in \mathbb{P}(Q \times V \times Q)$ is a transition relation,
- $E \in \mathbb{P}(Q \times Q)$ is an ε-transition relation
- $S \subseteq Q$ is a set of start states, and
- $F \subseteq Q$ is a set of final states.

□

```
class FA: virtual public FAabs {
        // Q is a finite set of states
        StatePool Q;
        // S is a set of start states, F is a
            set of final states
        StateSet S, F;
        // Transitions maps each State to its
            out-transitions.
        TransRel Transitions;
```

```
        // E is the epsilon transition
           relation.
        StateRel E;
}
```

StatePool:All states in an automaton are allocated from a StatePool. StatePool's can be merged together to form a larger one. (Care must be taken to rename any relations or sets (during merging) that depend on the one StatePool.) State is in [0,next)

```
class StatePool {
        int next; // The next one to be
           allocated.
}
```

StateSet:The StateSet is normally associated (implicitly) with a particular StatePool; whenever a StuteSet is to interact with another (from a different StatePool), its States must be renamed (to avoid a name clash). The capacity of a StateSet must be explicitly managed; many set operations are not bounds-checked when assert() is turned off.

```
class StateSet :protected BitVec {
        // How many States can this set
           contain?
        // [O, domain()) can be contained in
           *this.
        inline int domain() const;

        // set How many States can this set
           contain.
        // [O, r) can be contained in *this.
        inline void set_domain(const int r);
}
class BitVec {
```

```
            // uesd max number bits in data,
               denote width(domain),[0,
               bits_in_use) ==> [0,width)
            int bits_in_use;
            // number of words, 1,2,3,...
            int words;
            // save bytes of words,[0,1,2,...
               width(domain)]
            unsigned int *data;
    }
```

transition relation: $T \in Q \to \mathbb{P}(V \times Q), T(p) = \{(a,q)|(p,a,q) \in T\}$, 表示状态 p 的 out-transitions. see Fig 1.1

```
    // V --> Q
    struct TransPair {
    CharRange transition_label;
    State transition_destination;
    }
    class TransImpl { TransPair *data; }
    class Trans:protected TransIml { }

    // map: state(r) --> (T=Trans) out-
       transitions of r
    // SteteTo::data[r] = out-transitions of
       state r
    class TransRel:public StateTo<Trans> {}

    // map: state(r) --> T
    // data[r] = T
    template <class T> class SteteTo {
    T *data; // 动态数组的index(即状态的index)状
       态的out-transitions
    }
```

```
class FA:virtual public FAabs {
TransRel Transitions;// maps each State to
    its out-transitions.
}
```

ε-relation: $E \in \mathbb{P}(Q \times Q) \Rightarrow E \in Q \to \mathbb{P}(Q), E(p) = \{q|(p,q) \in E\}$, 表示 ε 连接状态 p 和状态 q.

```
// Implement binary relations on States. This
     is most often used for epsilon
    transitions.
// map: state(r) —> {StateSet}
// StateTo::data[r] = {StateSet}, 表示状态r与
    {StateSet}的二元关系
class StateRel :protected StateTo<StateSet> {
}

class FA:virtual public FAabs {
// E is the epsilon transition relation.
StateRel E;
}
```

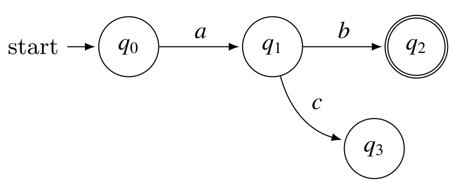


图 1.1: q_1 in-transition: $\{(q_0,a,q_1)\}$; q_1 out-transition: $\{(q_1,b,q_2),(q_1,c,q_3)\}$

[WATSON93a, p6] **the signatures of the transition relations:**

$T \in \mathbb{P}(Q \times V \times Q)$

$T \in V \to \mathbb{P}(Q \times Q)$

$T \in Q \times Q \to \mathbb{P}(V)$

$T \in Q \times V \to \mathbb{P}(Q)$

$T \in Q \to \mathbb{P}(V \times Q)$

for example, the function $T \in Q \to \mathbb{P}(V \times Q)$ is defined as $T(p) = \{(a,q) : (p,a,q) \in T\}$

$T \in \mathbb{P}(Q \times V \times Q), T = \{(s,a,q)\}$
$T(p) \in Q \to \mathbb{P}(V \times Q), T(p) = \{(a,q) : (s,a,q) \in T\}$

$p,q \in Q, a \in V$
$T : Q \times V \to \mathbb{P}(Q)$
$T(p,a) = \{q\}$

$Q_{map} : \mathbb{P}(Q \times V), Q_{map} = \{(q,a) : (s,a,q) \in T\}$
$Q_{map}(q) = \{a : (s,a,q) \in T\}$
${Q_{map}}^{-1} : V \nrightarrow \mathbb{P}(Q), {Q_{map}}^{-1} = \{(a,q) : (s,a,q) \in T\}$

1.2 Properties of finite automata

$$M = (Q,V,T,E,S,F), M_0 = (Q_0,V_0,T_0,E_0,S_0,F_0), M_1 = (Q_1,V_1,T_1,E_1,S_1,F_1)$$

Definition 1.2 (Size of an FA). Define the size of an FA as $|M| = |Q|$

Definition 1.3 (Isomorphism 同构 ($\cong$) of $FA's$). We define isomorphism ($\cong$) as an equivalence relation on FA's. M_0 and M_1 are isomorphic (written $M_0 \cong M_1$) if and only if $V_0 = V_1$ and there exists a bijection 双射 $g \in Q_0 \to Q_1$ such that

- $T_1 = \{(g(p),a,g(q)) | (p,a,q) \in T_0\}$
- $E_1 = \{(g(p),g(q)) | (p,q) \in E_0\}$
- $S_1 = \{g(s) | s \in S_0\}$ and
- $F_1 = \{g(f) | f \in F_0\}$

(see Fig 1.2). □

Definition 1.4 (Extending the transition relation T). We extend transition relation $T \in V \to \mathbb{P}(Q \times Q)$ to $T^* \in V^* \to \mathbb{P}(Q \times Q)$ as follows:

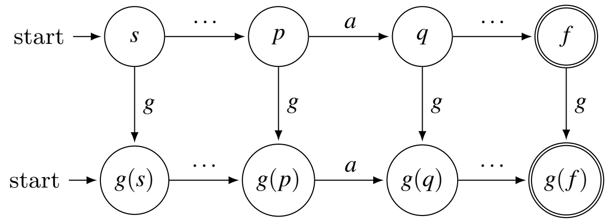


图 1.2: Isomorphism$M_0 \cong M_1$ if and only if $V_0 = V_1$ and there exists a bijection $g \in Q_0 \to Q_1$

$T^*(\varepsilon) = E^*$

and (for $a \in V, w \in V^*$)

$T^*(aw) = E^* \circ T(a) \circ T^*(w)$

Operator $\circ$ (composition is defined in Convention 1).

This definition could also have been presented symmetrically. $\square$

Note 1.1. $s_1, s_2, s_3, s_4 \in Q, a \in V, w \in V^*$

$E = T(\varepsilon) = \{(s_1, s_2)\}, T(a) = \{(s_2, s_3)\}, T^*(w) = \{(s_3, s_4)\}$

$$T^*(aw) = E^* \circ T(a) \circ T^*(w)$$
$$= \{(s_1, s_2)\} \circ \{(s_2, s_3)\} \circ \{(s_3, s_4)\} \quad = \{(s_1, s_3)\} \circ \{(s_3, s_4)\} = \{(s_1, s_4)\}$$

Note 1.2. $T \in Q \times V \to \mathbb{P}(Q)$, extend to: $T^* \in Q \times V^* \to \mathbb{P}(Q)$

$\forall q \in Q, w \in V^*, a \in V$，

1. $T^*(q, \varepsilon) = q$
2. $T^*(q, wa) = T(T^*(q, w), a)$

$$\begin{aligned} T^*(q, a) &= T^*(q, \varepsilon a) \\ &= T(T^*(q, \varepsilon), a) \\ &= T(q, a) \end{aligned}$$

两值相同，不用区分这两个符号。

Convention 1 (Relation composition) *Given sets A, B, C (not necessarily different) and two relations, $E \subseteq A \times B$ and $F \subseteq B \times C$, we define relation composition (infix operator* 中缀操作符 $\circ$*) as:*

$E \circ F = \{(a, c) | (\exists b \in B), (a, b) \in E \wedge (b, c) \in F)\}$ $\square$

Note 1.3. if $\exists b \in B, (a,b) \in E, (b,c) \in F$, then

$E : A \to B \Rightarrow E(a) = b$

$F : B \to C \Rightarrow F(b) = c$

$E \circ F = \{(a,b)\} \circ \{(b,c)\} = \{a,c\}$

$$\begin{aligned}(E \circ F)(a) &= F(E(a)) \\ &= F(b) = c\end{aligned}$$

$$\begin{tikzcd} A \arrow[r, "E"] \arrow[dr, "f"'] & B \arrow[d, "F"] \\ & C \end{tikzcd}$$

图 1.3: $E \circ F = (F \circ E)(a) = F(E(a)) = c = f(a)$

Remark 1.1. We also sometimes use the signature $T^* \in Q \times Q \to \mathbb{P}(V^*)$ □

Note 1.4. $T(p,q) = \{w | p,q \in Q, w \in V^*\}$

Remark 1.2. if $E = \emptyset$ then $E^* = \emptyset^* = I_Q$ where I_Q is the identity relation 单位关系 on the states of M.

Definition 1.5 (The language between states). The language between any two states $q_0, q_1 \in Q$ is $T^*(q_0, q_1)$. □

Definition 1.6 (Left and right languages). The left language of a state (in M) is given by function, $\overleftarrow{L}_M \in Q \to \mathbb{P}(V^*)$,where

$\overleftarrow{L}_M(q) = (\cup s : s \in S : T^*(s,q))$

The right language of a state (in M) is given by function $\overrightarrow{L}_M \in Q \to \mathbb{P}(V^*)$, where

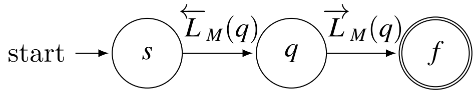
$\overrightarrow{L}_M(q) = (\cup f : f \in F : T^*(q,f))$

The subscript M is usually dropped when no ambiguity can arise. □

Example 1.1. $T^* \in Q \times Q \to \mathbb{P}(V^*), \overleftarrow{L}_M, \overrightarrow{L}_M \in Q \to \mathbb{P}(V^*)$.

$\overleftarrow{L}_M(q)$ ={能引导 M 从开始状态到达 q 状态的字符串集合}, (从 q 往左看)

$\overrightarrow{L}_M(q)$ ={能引导 M 从开始状态到达 q 状态的字符串集合}, (从 q 往右看)



see Fig 1.4.

$$
\begin{aligned}
\overleftarrow{L}_M(q_2) &= (s \to q_1 \to q_2) \cup (s \to (q_1 \to q_3)^* \to q_1 \to q_2) \cup (s \to (q_1 \to q_3)^* \to q_3 \to q_2) \\
&= [(s \to q_1 \to q_2) \cup (s \to (q_1 \to q_3)^* \to q_1 \to q_2)] \cup (s \to (q_1 \to q_3)^* \to q_3 \to q_2) \\
&= (s \to (q_1 \to q_3)^* \to q_1 \to q_2) \cup (s \to (q_1 \to q_3)^* \to q_3 \to q_2) \\
&= \{1(10)^*0, 1(10)^*1\} \\
\overrightarrow{L}_M(q_2) &= \{01^*0, 10^*1(001^*0 + (10)^*1)\}
\end{aligned}
$$

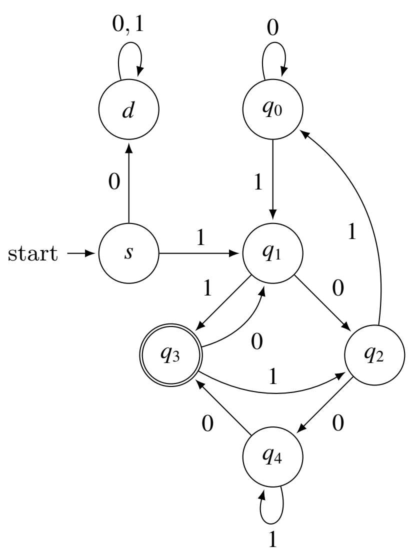


图 1.4: $\{x | x \in \{0,1\}^+$ 且当把 x 看成二进制数时，x 模 5 与 3 同余，要求当 x 为 0 时，$|x| = 1$, 且当 $x \neq 0$ 时，x 的首字符为 $1\}$ 语言对应的 *DFA*

Definition 1.7 (Language of an *FA*). The language of a finite automaton (with alphabet V) is given by the function $L_{FA} \to \mathbb{P}(V^*)$ defined as:

$L_{FA}(M) = (\cup s, f : s \in S \wedge f \in F : T^*(s,f))$ (所有从开始状态到接受状态的字符串集合) □

Property 1.1 (Language of an FA). From the definition of left and right languages (of a state), we can also write:

$L_{FA}(M) = (\cup f : f \in F : \overleftarrow{L}(f))$ (所有从 s 到 f 的字符串集合，从 f 向左看)

and

$L_{FA}(M) = (\cup s : s \in S : \overrightarrow{L}(s))$ (所有从 s 到 f 的字符串集合，从 s 向右看) □

Definition 1.8 (ε-*free* 无 ε 转移). Automaton M is ε-*free* if and only if $E = \emptyset$. □

Remark 1.3. Even if M is ε-*free* it is still possible that $\varepsilon \in L_{FA}(M) : inthiscase S \cap F \neq \emptyset$. (开始状态也是接受状态) □

Form [WATSON93a, Convention A.4] (Tuple projection).

Convention 2 (Tuple projection) *For an n-tuple $t = (x_1, x_2, \ldots, x_n)$ we use the notation $\pi_i(t)(1 \leq i \leq n)$ to denote tuple element x_i; we use the notation $\bar{\pi}_i(t)(1 \leq i \leq n)$ to denote the $(n-1)$-tuple $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. Both π and $\bar{\pi}$ extend naturally to sets of tuples.* □

Form [WATSON93a, Definition A.20] (Tuple and relation reversal).

Definition 1.9 (Tuple and relation reversal). For an n-tuple $(x_1, x_2, \ldots, x_n)$ define reversal as (postfix and superscript) function R:

$(x_1, x_2, \ldots, x_n)^R = (x_n, x_n - 1, \ldots, x_2, x_1)$

Given a set A of tuples, we define $A^R = \{x^R : x \in A\}$. □

Definition 1.10 (Reachable states). For M we can define a reachability relation $Reach(M) \subseteq (Q \times Q)$ defined as

$Reach(M) = (\bar{\pi}_2(T) \cup E)^*$ see[1]

Functions π and $\bar{\pi}$ are defined in Convention 2. Similarly the set of start-reachable states is defined to be:

[1] $\{(p_1, q_1), (p_2, q_2), \ldots\}$

$SReachable(M) = Reach(M)(S)$ see[2]

and the set of final-reachable states is defined to be:

$FReachable(M) = (Reach(M))^R(F)$ see[3]

Reversal of a relation is defined in Definition 1.9. The set of useful states is: $Reachable(M) = SReachable(M) \cap FReachable(M)$ □

Remark 1.4. For *FA* $M = (Q,V,T,E,S,F)$,function *SReachable* satisfies the following interesting property:

$q \in SReachable(M) \equiv \overleftarrow{L}_M(q) \neq \emptyset$

FReachable satisfies a similar property:

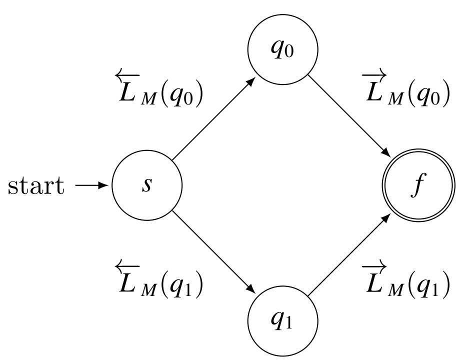
$q \in FReachable(M) \equiv \overrightarrow{L}_M(q) \neq \emptyset$ □

Example 1.2. $T \in \mathbb{P}(Q \times V \times Q), T = \{(p,a,q)|p,q \in Q, a \in V\}$,

$\bar{\pi}_2(T) = \{(p,q)|(p,a,q) \in T\}$

$Q_{map} = (\bar{\pi}_1(T))^R, Q_{map} = \{(a,q)|(p,a,q) \in T\}^R = \{(q,a)|(p,a,q) \in T\}$

□



e.g. $p = \{1,2\} \in Q_1 \subseteq \mathbb{P}(Q_0), \overrightarrow{L}_{M_1}(p) = \overrightarrow{L}_{M_0}(1) \cup \overrightarrow{L}_{M_0}(2)$

[2] 从 start state 可以到达的状态集合

[3] 可以到达 final state 的状态集合

$$
\begin{tikzcd}
Q \arrow[r, "subset"] \arrow[dr, "f"'] & P(Q) \arrow[d, "useful_s"] \\
& D
\end{tikzcd}
$$

图 1.5: $subset \circ usefus_s = usefus_s(subset(Q,V,\emptyset,S,F)) = (D,V,T',\emptyset,S',F')$

1.3 Σ-algebras and regular expressions

Σ-homomorphism

X 集合中的元素与有序集 S 中的元素一一对应，称 X 是 S-sorted.
$S = \{1,3,7,9\}, X = \{d,a,c,f\}, s \in S, X_s \in X$
S 是有序的,$S_{s_1} = 1, S_{s_2} = 3, S_{s_3} = 7, S_{s_4} = 9$
X 与 S 中的元素一一对应。$X_{s_1} = d, X_{s_2} = a, X_{s_3} = c, X_{s_4} = f$

Σ-homomorphism 同态: $(V,F) \Leftrightarrow (W,G)$, 载体 (V,W) 和操作 (F,G) 一一对应。
Σ-homomorphism function: $h \in V \to W$
$L(v) = (h \circ f)(v) = h(f(v)) = g(w) = L_{reg} = L_V = L_W$
$L(v) = (g \circ h)(v) = g(h(v)) = g(w) = L_{reg} = L_V = L_W$
$\Rightarrow h(f(v)) = g(h(v))$

$$
\begin{tikzcd}
V \arrow[r, "h"] \arrow[d, "f"'] \arrow[dr, "L"] & h(v) \arrow[d, "g"] \\
f(v) \arrow[r, "h"] & g(h(v))
\end{tikzcd}
$$

图 1.6: $(h \circ f)(v) = (g \circ h)(v) \Rightarrow h(f(v)) = g(h(v))$

Example 1.3. $\Sigma = (S, \Gamma)$, sort: $expr$, $\Gamma := \{a, plus\}, a$ is a constant. operator $plus : expr \times expr \to expr$.
$\Sigma-$term algebra: $plus[a,a], plus[plus[a, plus[a,a]], a]$
$\Sigma-$algebra X,carrier set: natural number, constant 0. operator $f_{plus}(x,y) =$

$$\begin{tikzcd}
(e_1,e_2) \arrow[r, "h"] \arrow[d, "f"'] \arrow[dr, "L"] & (h(e_1),h(e_2)) \arrow[d, "g"] \\
f(e_1,e_2)\circ \arrow[r, "h"] & g(h(e_1),h(e_2))
\end{tikzcd}$$

图 1.7: $(h \circ f)(e_1,e_2) = (g \circ h)(e_1,e_2) \Rightarrow h(f(e_1,e_2)) = g(h(e_1),h(e_2))$

$(x \quad max \quad y)+1$

Σ-homomorphism function(”expression tree height”): $h_e xpr : \Sigma-$term algebra $\rightarrow X$

$$\begin{aligned}
(h_{expr} \circ plus)(s) &= (f_{plus} \circ h_{expr})(s) \\
h_{expr}(plus(s)) &= f_{plus}(h_{expr}(s)) \\
left : s \leftarrow e, f &\quad \Rightarrow \quad plus[e,f] \\
right : s \leftarrow e, f &\quad \Rightarrow \quad f_{plus}(h_{expr}(e), h_{expr}(f) \\
h_{expr}(plus(e,f)) &= f_{plus}(h_{expr}(e), h_{expr}(f)) \\
&= (h_{expr}(e) \quad max \quad h_{expr}(f)) + 1 \\
and, & \\
h_{expr}(a) &= 0
\end{aligned}$$

$$\begin{tikzcd}
S \arrow[r, "h_{expr}"] \arrow[d, "plus"'] \arrow[dr, "L"] & X \arrow[d, "f_{plus}"] \\
\circ \arrow[r, "h_{expr}"] & \circ
\end{tikzcd}$$

图 1.8: $(h_{expr} \circ plus)(s) = (f_{plus} \circ h_{expr})(s) \Rightarrow h_{expr}(plus(s)) = f_{plus}(h_{expr}(s))$

Definition 1.11 (Regular expressions). We define regular expressions (over alphabet V) as the $\Sigma-$term algebra over signature $\Sigma = (S, O)$ where

- S consists of a single sort Reg (for regular expression), and

$$\begin{tikzcd}
(e,f) \arrow[r, "h_{expr}"] \arrow[d, "plus"'] \arrow[dr, "L"] & (h_{expr}(e),h_{expr}(f)) \arrow[d, "f_{plus}"] \\
plus[e,f] \arrow[r, "h_{expr}"] & f_{plus}(h_{expr}(e),h_{expr}(f))
\end{tikzcd}$$

图 1.9: $(h_{expr}\circ plus)(e,f) = (f_{plus}\circ h_{expr})(e,f) \Rightarrow h_{expr}(plus[e,f]) = f_{plus}(h_{expr}(e),h_{expr}(f))$

- O is a set of several constans: $\varepsilon,\emptyset,a_1,a_2,\ldots,a_n;\ Reg$ (where $V=\{a_1,a_2,\ldots,a_n\}$) and five operators $.: Reg\times Reg\to Reg$ (the dot operator), $\cup: Reg\times Reg\to Reg, *: Reg\to Reg, +: Reg\to Reg,$ and $?: Reg\to Reg$.

$V := RE$(正则表达式), $W := FA$(有限自动机)

Σ-homomorphism function: $f\in RE\to FA$

$F: RE_{opt}$ 运算，二元: union(or),concat; 一元: star,plus,question;

常量:epsilon,empty,symbol

$G: FA_{opt}$ 运算, 同上

$$\begin{tikzcd}
RE \arrow[r, "f"] \arrow[dr, "L_{RE}"'] & FA \arrow[d, "L_{FA}"] \\
& L_{reg}
\end{tikzcd}
\qquad
\begin{tikzcd}
(re_1,re_2) \arrow[r, "f"] \arrow[d, "RE_{opt}"'] \arrow[dr] & (f(re_1),f(re_2)) \arrow[d, "FA_{opt}"] \\
RE_{opt}(re_1,re_2) \arrow[r, "f"] & FA_{opt}(f(re_1),f(re_2))
\end{tikzcd}$$

```
//Sigma.h
template<class T>
class Reg :public T {
// Helper for constructing the homomorphic
    image of a regular expression.
// T is carrier set: RE,FA,RFA，
// 各自的操作，分别在Sig-RE.cpp,Sig-FA.cpp,
    Sig-RFA.cpp中定义
inline void homomorphic_image(const RE& r);
Reg<T>& epsilon();
Reg<T>& empty();
Reg<T>& symbol(const CharRange r);
Reg<T>& Or(const Reg<T>& r);
```

```
    Reg<T>& concat(const Reg<T>& r);
    Reg<T>& star();
    Reg<T>& plus();
    Reg<T>& question();
    }
```

Definition 1.12 (The *nullable* $\Sigma-$algebra). We define the *nullable* $\Sigma-$algebra as follows:

- The carrier set is $\{true, false\}$.
- $a \in V, E_1, E_2 \in RE, \varepsilon \in E_1^*, \varepsilon \in E_1^?, \varepsilon \notin E_1^+$

$$
\begin{aligned}
nullable(\varepsilon) &= true \\
nullable(\emptyset) &= nullable(a) = false \\
nullable(E_1 \vee E_2) &= nullable(E_1 \cup E_2) \\
nullable(E_1 \wedge E_2) &= nullable(E_1 \cdot E_2) \\
nullable(E_1^*) &= true \\
nullable(E_1^+) &= nullable(E_1) \\
nullable(E_1^?) &= true \\
nullable(E_1) &= \begin{cases} true & \varepsilon \in E_1 \\ false & \varepsilon \notin E_1 \end{cases}
\end{aligned}
$$

1.4 Constructing ε-lookahead automata

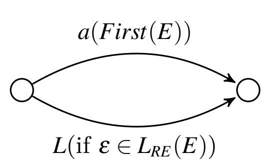


图 1.10: Lookahead function:$look(E,L) = First \cup$ if $(Null(E))$ then L else $\emptyset$ fi

1.5 Others

Definition 1.13 (Prefix-closure[Chrison2007]). Let $L \subseteq V^*$,then

$$\overline{L} := \{s \in V^* : (\exists t \in V^*)[st \in L]\}$$

In words, the prefix closure of L is the language denoted by $\overline{L}$ and consisting of all the prefixes in L. In general, $L \subseteq \overline{L}$.

L is said to be prefix-closed if $L = \overline{L}$. Thus language L is prefix-closed if any prefix of any string in L is also an element of L.

$L_1 = \{\varepsilon, a, aa\}, L_1 = \overline{L_1}, L_1$ is prefix-closed.

$L_2 = \{a, b, ab\}, \overline{L_2} = \{\varepsilon, a, b, ab\}, L_2 \subset \overline{L_2}, L_2$ is not prefix closed.

Definition 1.14 (Post-closure[Chrison2007]). Let $L \subseteq V^*$ and $s \in L$. Then the post-language of L after s, denoted by L/s, is the language

$$L/s := \{t \in V^* : st \in L\}$$

By definition, $L/s = \emptyset$ if $s \notin \overline{L}$.

Definition 1.15 (Left derivatives[WATSON93a]). Given language $A \subseteq V^*$ and $w \in V^*$ we define the left derivative of A with respect to w as:

$$w^{-1}A = \{x \in V^* : wx \in A\}$$

A 关于 w 的左导数，就是 A 中：$\{w$ 的后缀组成的字符串集合$\}$。

Sometimes derivatives are written as $D_w A$ or as $\frac{dA}{dw}$. Right derivatives are analogously defined. Derivatives can also be extended to $B^{-1}A$ where B is also a language.

Example 1.4. $A = \{a, aab, baa\}, a^{-1}A = D_a A = \frac{dA}{da} = \{\varepsilon, ab, \emptyset\} = \{\varepsilon, ab\}$ □

Example 1.5. $L = \{ba, baa, baab, ca\}, w = \{ba\}$,

$w^{-1}L = \{\varepsilon, a, ab, \emptyset\} = \{\varepsilon, a, ab\}$

$(wa)^{-1}L = (baa)^{-1}L = \{\emptyset, \varepsilon, b, \emptyset\} = \{\varepsilon, b\}$

$a^{-1}(w^{-1}L) = a^{-1}\{\varepsilon, a, ab\} = \{\emptyset, \varepsilon, b\} = \{\varepsilon, b\}$

$w \in L \equiv \varepsilon \in w^{-1}L, and (wa)^{-1}L = a^{-1}(w^{-1}L)$ □

Example 1.6. $a^{-1}\{a\} = \{\varepsilon\}; \quad a^{-1}\{b\} = \emptyset, \quad \Leftarrow if(a \neq b)$ □

Example 1.7. $L_0 = \{ab\}, L_1 = \{ac\}, L_0L_1 = \{abac\}$

$a^{-1}(L_0L_1) = \{bac\}$

$a^{-1}(L_0L_1) = (a^{-1}L_0)L_1 \cup \emptyset \quad \Leftarrow (\varepsilon \notin L_0)$

$= \{b\}L_1 = \{bac\}$ □

Example 1.8. $L_0 = \{\varepsilon, ab\}, L_1 = \{ac\}, L_0L_1 = \{ac, abac\}$

$a^{-1}(L_0L_1) = \{c, bac\}$

$a^{-1}(L_0L_1) = (a^{-1}L_0)L_1 \cup a^{-1}L_1 \quad \Leftarrow (\varepsilon \in L_0)$

$= \{\emptyset, b\}L_1 \cup \{c\} = \{c, bac\}$ □

证明. $a^{-1}(L_0L_1)$

$1.if(\varepsilon \in L_0) \Rightarrow a^{-1}(L_0L_1) = (a^{-1}L_0)L_1 \cup a^{-1}L_1$

$L_0 = (L_0 \backslash \{\varepsilon\}) \cup \{\varepsilon\}$

$a^{-1}(L_0L_1) = a^{-1}(((L_0 \backslash \{\varepsilon\}) \cup \{\varepsilon\})L_1)$

$= a^{-1}(L_0L_1 \cup L_1)$

$a^{-1}L_0 = a^{-1}((L_0 \backslash \{\varepsilon\}) \cup \{\varepsilon\})$

$= a^{-1}(L_0 \ \backslash \{\varepsilon\}) \cup a^{-1}\{\varepsilon\}$

$= a^{-1}L_0 \cup \emptyset = a^{-1}L_0$

From [Hopcroft2008, p99]

(1) 如果 L 是一个语言，a 是一个符号，则 L/a(称作 L 和 a 的商) 是所有满足如下条件的串 w 的集合：wa 属于 L。例如，如果 $L=\{a,aab,baa\}$, 则 $L/a=\{\varepsilon,ba\}$, 证明：如果 L 是正则的，那么 L/a 也是。提示：从 L 的 DFA 出发，考虑接受状态的集合。

(2) 如果 L 是一个语言，a 是一个符号，则 $a\backslash L$ 是所有满足如下条件的串 w 的集合: aw 属于 L。例如，如果 $L=\{a,aab,baa\}$, 则 $a\backslash L=\{\varepsilon,ab\}$, 证明：如果 L 是正则的，那么 $a\backslash L$ 也是。提示：记得正则语言在反转运算下是封闭的，又由 (1) 知，正则语言的商运算下是封闭的。

Definition 1.16 (Kleene-closure[Chrison2007]). Let $L\subseteq V^*$, then

$$L^* := \{\varepsilon\}\cup L\cup LL\cup LLL\cup\cdots$$

This is the same operation that we defined above for the set V, except that now it is applied to set L whose elements may be strings of length greater than one. An element of L^* is formed by the concatenation of a finite (but possibly arbitrarily large) number of elements of L; this includes the concatenation of ”zero” elements, that is the empty string ε. Note that $*$ operation is idempotent: $(L^*)^*=L^*$.

$$\begin{aligned} L^* &= \{\varepsilon\}+L^+ \\ &= \{\varepsilon\}\cup(L\backslash\{\varepsilon\})L^* \\ &= \{\varepsilon\}+L+LL+LLL+\cdots \end{aligned}$$

1.6 Linear equation

see [Jean2018, 5.3,p64].

We give an algorithm to covert an automaton to a rational(regular) expression. The algorithm amounts to solving a system of linear equations on languages. We first consider an equation of the form

$$X=KX+L \tag{1.1}$$

Proposition 1.1 (Arden’s Lemma). *if K does not contain the empty word, then $X=K^{\star}L$ is the unique solution of the equation $X=KX+L$.*

where K and L are languages and X is the unknown. When K does not contain the empty word, the equation admits a unique solution.

证明. Replacing X by K^*L in the expression $KX+L$,one gets

$$K(K^*)L+L=K^+L+L=(K^+L+L)=K^*L,$$

and hence $X=K^*L$ is a solution of (1.1). see[1]

To Prove uniqueness, consider two solutions X_1 and X_2 of (1.1). By symmetry,it suffices to show that each word u of X_1 also belongs to X_2. Let us prove this result by induction on the length of u.

If $|u|=0$, u is the empty word[2] and if $u\in X_1=KX_1+L$, then necessarily $u\in L$ since $\varepsilon\notin K$. But in this case, $u\in KX_2+L=X_2$. see[3]

For the induction step, consider a word u of X_1 of length $n+1$. Since $X_1=KX_1+L, u$ belongs either to L or to KX_1. if $u\in L$, then $u\in KX_2+L=X_2$. If $u\in KX_1$ then $u=kx$ for some $k\in K$ and $x\in X_1$. Since k is not the empty word, one has necessarily $|x|\leq n$ and hence by induction $x\in X_2$.[see[4]] It follows that $u\in KX_2$ and finally $u\in X_2$. This conclude the induction and the proof of the proposition. □

From [Wonham2018, p74] The *length* $|s|$ of a string $s\in\Sigma^*$ is defined according to

$$|\varepsilon|=0;|s|=k,\text{if } s=\sigma_1\sigma_2\cdots\sigma_k\in\Sigma^+$$

Thus $|cat(s,t)|=|s|+|t|$.

1

$$\begin{aligned}K^*&=\{\varepsilon\}+K^+\\&=\{\varepsilon\}+(K\backslash\{\varepsilon\})K^*\\&=\{\varepsilon\}+K+KK+KKK+\cdots\end{aligned}$$

2 The empty word $=\varepsilon,|\varepsilon|=0$; if a language $M=\{\varepsilon\},|M|=1$, The empty language $M=\emptyset,|M|=0$. 文献 [Jean2018] 用 1 表示 ε, 因为 $\varepsilon K=K\varepsilon=K$, 因此,$\varepsilon$ 是连接运算的单位元，正是 1 表示的用意。0 表示 $\emptyset$, 它是并运算的单位元,$K\cup\emptyset=\emptyset\cup K=K$.

3 In this case,$|u|=0,X=\{\varepsilon\},|X|=1.$ *i.e.* $\varepsilon=K\varepsilon+L,\varepsilon=K+L$

4 $u=kx,|u|=|kx|=n+1,\varepsilon\notin K,|k|\geq 1,|x|\leq n$, 由假设知，$u$ 属于 X_1, 归纳 $|x|=0,|x|=1,\cdots,n,x\in X_2$.

A *language* over Σ is any subset of Σ^*, i.e. an element of the power set $Pwr(\Sigma^*)$; thus the definition includes both the empty language $\emptyset$, and Σ^* itself.

Note the distinction between $\emptyset$ (the language with no strings) and ε (the string with no symbols). For instance the language $\{\varepsilon\}$ is nonempty, but contains only the empty string.

From [Wonham2018, p78]

Proposition 1.2 ([Wonham2018]).

1. *If* $L = M^*N$ *then* $L = ML + N$
2. *If* $\varepsilon \notin M$ *then* $L = ML + N$ *implies* $L = M^*N$ □

Part(2) is Known as Arden's rule. Taken with Part(1) it says that if $\varepsilon \notin M$ then $L = M^*N$ is the unique solution of $L = ML + N$; in particular if $L = ML$ (with $\varepsilon \notin M$) then $L = \emptyset$

Exercise 1.1. Show by counterexample that the restriction $\varepsilon \notin M$ in Arden's rule cannot be dropped.

Solution 1.1. Examples text goes here.

Exercise 1.2. Prove Arden's rule. Hint: If $L = ML + N$ then for every $k \geq 0$

$$L = M^{k+1}L + (M^k + M^{k-1} + \cdots + M + \varepsilon)N$$

Solution 1.2.

Preliminaries :

$$
\begin{aligned}
M^* &= M^k + M^{k-1} + \cdots + M^1 + M^0 \qquad (k \geq 0) \\
&= M^k + M^{k-1} + \cdots + M^1 + \varepsilon \\
&= M^+ + \varepsilon \\
&= MM^* + \varepsilon \\
&= (M \backslash \{\varepsilon\})M^* + \varepsilon \\
M^+ &= M^k + M^{k-1} + \cdots + M^1 \qquad (k > 0) \\
&= M(M^k + M^{k-1} + \cdots + M^1 + M^0) \\
&= MM^* \\
M^0 &= \{\varepsilon\} = 1 \\
M\varepsilon &= \varepsilon M = M \\
\varepsilon + \varepsilon &= \varepsilon \\
M + M &= M
\end{aligned}
$$

证明.

$$L = ML + N \Rightarrow$$

$$M^0L = M^1L + M^0N \tag{1.2}$$
$$M^1L = M^2L + M^1N \tag{1.3}$$
$$M^2L = M^3L + M^2N \tag{1.4}$$
$$\tag{1.5}$$

$$\cdots$$

$\Rightarrow$

$(M^0 + M^1 + M^2 + \cdots)L = (M^1 + M^2 + M^3 + \cdots)L + (M^0 + M^1 + M^2 + \cdots)N$

$\Rightarrow$

so,if $L = ML + N$,then for every $k \geq 0$

$L = M^{k+1}L + (M^k + M^{k-1} + \cdots + M + M^0)N$

$\Rightarrow$

$$L = M^{k+1}L + (M^k + M^{k-1} + \cdots + M + \varepsilon)N \tag{1.6}$$

(1) $k=0$

$L = ML + (\varepsilon)N = ML + N$

$\Rightarrow (1-M)L = N$

$(\varepsilon - M)L = N$

由于 $\varepsilon \notin M$, 左端不会消去 $\{\varepsilon\}$. 因此, 只能在 N 中找 L, 仅有唯一解: $L = \{\varepsilon\} = \{\text{empty word}\} \subseteq N$.

From [R.Su and Wonham2004, definition 2.3]

Definition 1.17. Let

$G_A = (X_A, \Sigma, \xi_A, x_{A,0}, X_{A,m})$

$G_A = (X_B, \Sigma, \xi_B, x_{B,0}, X_{B,m})$

G_B is a DES-epimorphic image(满射像) of G_A under DES-epimorphism $\theta : X_A \to X_B$ if

1. $\theta : X_A \to X_B$ is surjective(满射)
2. $\theta(x_{A,0}) = x_{B,0}$ and $\theta(X_{A,m}) = X_{B,m}$
3. $(\forall x \in X_A)(\forall \sigma \in \Sigma)\xi_A(x,\sigma)! \Rightarrow [\xi_B(\theta(x),\sigma)!\&\xi_B(\theta(x),\sigma) = \theta(\xi_A(x,\theta))]$
4. $(\forall x \in X_B)(\forall \sigma \in \Sigma)\xi_B(x,\sigma)! \Rightarrow [(\exists x' \in X_A)\xi_A(x',\sigma)!\&\theta(x') = x]$

In particular,G_B is DES-isomorphic(同构) to G_A if $\theta : X_A \to X_B$ is bijective(双射).

see figure 1.11.

$\theta(x_{A,0}) = x_{B,0}$ and $\theta(X_{A,m}) = X_{B,m}$

$\theta(x_A) = x_B$ and $\theta(x'_A) = x'_B$

$\xi_A(x_A,\sigma) = x'_A$ and $\xi_B(x_B,\sigma) = x'_B \Rightarrow$ definition 1.17 (3,4)

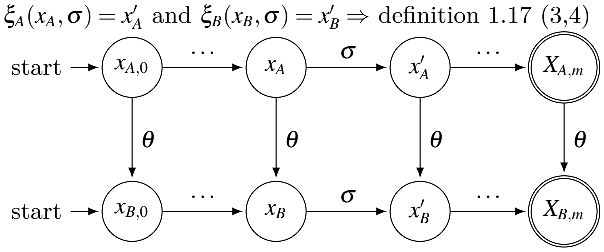


图 1.11: definition 1.17, G_B is a DES-epimorphic image(满射像) of G_A under DES-epimorphism $\theta : X_A \to X_B$

References

[Hopcroft2008] John E. Hopcroft,Rajeev Motwani,Jeffrey D. Ullman 著, 孙家骕等译, 自动机理论、语言和计算机导论,Third Edition, 机械工业出版社,2008.7

[WATSON93a] WATSON, B. W. *A taxonomy of finite automata construction algorithms*, Computing Science Note 93/43, Eindhoven University of Technology, The Netherlands, 1993. Available by ftp from ftp.win.tue.nl in pub/techreports/pi.

[WATSON93b] WATSON, B. W. *A taxonomy of finite automata minimization algorithms*, Computing Science Note 93/44, Eindhoven University of Technology, The Netherlands, 1993. Available by ftp from ftp.win.tue.nl in pub/techreports/pi.

[WATSON94a] WATSON, B. W. *An introduction to the FIRE engine: A C++ toolkit for FInite automata and Regular Expressions*, Computing Science Note 94/21, Eindhoven University of Technology, The Netherlands, 1994. Available by ftp from ftp.win.tue.nl in pub/techreports/pi

[WATSON94b] WATSON, B.W. *The design. and implementation of the FIRE engine: A C++ toolkit for FInite automata and Regular Expressions*, Computing Science Note 94/22, Eindhoven University of Technology, The Netherlands, 1994. Available by ftp from ftp.win.tue.nl in pub/techreports/pi.

[Chrison2007] Christos G. Cassandras and Stéphane Lafortune, *Introduction to Discrete Event Systems*,Second Edition,New York,Springer,2007

[Wonham2018] W. M. Wonham and Kai Cai,*Supervisory Control of Discrete-Event Systems*, Revised 2018.01.01

[Jean2018] Jean-Éric Pin, *Mathematical Foundations of Automata Theory*,Version of June 15,2018

[蒋宗礼 2013] 蒋宗礼, 姜守旭, *形式语言与自动机理论（第 3 版）*, 清华大学出版社,2013.05

[Lipschutz2007] S. Lipschutz and M. L. Lipson, *Schaum's Outline of Theory and Problems of Discrete Mathematics*, Third Edition, New York: McGraw-Hill, 2007.

[Rosen2007] K. H. Rosen, *Discrete Mathematics and Its Applications*, Seventh Edition, New York: McGraw-Hill, 2007.

[R.Su and Wonham2004] R. Su and W. M. Wonham, *Supervisor reduction for discrete-event systems*, Discrete Event Dyn. Syst., vol. 14, no. 1, pp. 31–53, Jan. 2004.

1. S. Maclane and G. Birkhoff, *Algebra*, Third Edition, New York: Bchelsea Publishing Company, 1988.