目录

Chapter 1
Hopcroft's algorithm

1.0.1 algorithm

Member function min_Hopcroft implements Hopcroft's $n \log n$ minimization algorithm, as presented in [[WATSON94b], Algorithm 4.8].

The combination of the out-transitions of all of the States is stored in a **CRSet** C.

Set L from the abstract algorithm is implemented as a mapping from States to int (an array of int is used).

Array L should be interpreted as follows: if State q a representative, then the following pairs still require processing (are still in abstract set L):

$$([q], C_0), \cdots, ([q], C_{L(q)-1})$$

The remaining pairs do not require processing:

$$([q], C_{L[q])}), \cdots, ([q], C_{|C|-1})$$

This implementation facilitates quick scanning of L for the next valid State-CharRange pair.

1.0.2 Example

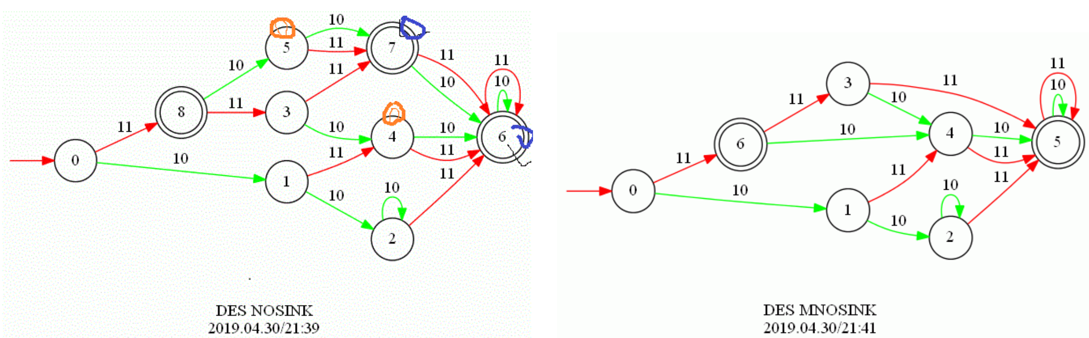
```
CRSet C;  // the out labels of State's: { 'a' 'b' }
int L[|Q|]: // the index of L = q: 对应等价类[q]; L[q]表示正在处理[q]对
    应的字符在C中的index,
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0 0

Initialize partitions, E_0:
{ 0 1 2 3 4 5 }
{ 6 7 8 }
```

Algorithm 1 Hopcroft's minimization algorithm

Input: $\mathscr{A} = (Q, i, F)$

Output: The equivalence classes of Q

$\hat{P} \leftarrow \{F, F^c\}$ ▷ The initial partition
$W \leftarrow \emptyset$ ▷ The waiting set
for all $a \in A$ **do**
  $ADD((min(F, F^c), a), W)$ ▷ initialization of the waiting set
end for
while $W \neq \emptyset$ **do**
  $(W, a) \leftarrow TakeSome(W)$ ▷ Take and remove some splitter
  for all $P \in \hat{P}$ **do** which is split by (W, a)
    $P', P'' \leftarrow (W, a)|P$ ▷ Compute the split
    Replace P by P' and P'' in $\hat{P}$ ▷ Refine the partition
  end for
  for all $b \in A$ **do** ▷ Update the waiting set
    if $(P, b) \in W$ **then**
      Replace (P, b) by (P', b) and (P'') in W
    else
      $ADD((min(P', P''), b), W)$
    end if
  end for
end while



(a) DFA (b) mini-DFA

图 1.1: example minimization

```
 Initialize L repr{F}:  { 6 }
L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 2 0 0

--- while each [q], (split [p] w.r.t ([q],a))
L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 2 0 0
```

```
Pick one [q] in L, Processing [q]= index of L = [6], 'b'
current all partitions:
{ 0  1  2  3  4  5 }
{ 6  7  8 }

=== for each [p], (split [p] w.r.t (index of L)[6],'b')

===split[0] w.r.t (index of L)[6],'b')
before split, partitions:
{ 0  1  2  3  4  5 }
{ 6  7  8 }

new split of [0] is [1]
after split, partitions:
{ 0  2  3  4  5 }
{ 1 }
{ 6  7  8 }

before L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 1 0 0

p and r are the new representatives. Now update L with the smallest of
    [0],[1]
using [r] = [1],L[r]=C.size();

affter L:
0 1 2 3 4 5 6 7 8
0 2 0 0 0 0 1 0 0

===split[6] w.r.t (index of L)[6],'b')
before split, partitions:
{ 0  2  3  4  5 }
{ 1 }
{ 6  7  8 }

new split of [6] is [8]
after split, partitions:
{ 0  2  3  4  5 }
{ 1 }
```

```
{ 6  7 }
{ 8 }

before L:
0 1 2 3 4 5 6 7 8
0 2 0 0 0 0 1 0 0

p and r are the new representatives. Now update L with the smallest of
    [6],[8]
using [r] = [8],L[r]=C.size();

affter L:
0 1 2 3 4 5 6 7 8
0 2 0 0 0 0 1 0 2

--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
0 2 0 0 0 0 1 0 2
Pick one [q] in L, Processing [q]= index of L = [1], 'b'
current all partitions(eq.classes) repr:
{ 0  1  6  8 }
current all partitions:
{ 0  2  3  4  5 }
{ 1 }
{ 6  7 }
{ 8 }

=== for each [p], (split [p] w.r.t (index of L)[1],'b')
===split[0] w.r.t (index of L)[1],'b')
before split, partitions:
StateEqRel
{ 0  2  3  4  5 }
{ 1 }
{ 6  7 }
{ 8 }

new split of [0] is [-1]
===split[1] w.r.t (index of L)[1],'b')
before split, partitions:
{ 0  2  3  4  5 }
```

```
{ 1 }
{ 6  7 }
{ 8 }

new split of [1] is [-1]
==split[6] w.r.t (index of L)[1],'b')
before split, partitions:
{ 0  2  3  4  5 }
{ 1 }
{ 6  7 }
{ 8 }

new split of [6] is [-1]
==split[8] w.r.t (index of L)[1],'b')
before split, partitions:
StateEqRel
{ 0  2  3  4  5 }
{ 1 }
{ 6  7 }
{ 8 }

new split of [8] is [-1]
--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
0 1 0 0 0 0 1 0 2
Pick one [q] in L, Processing [q]= index of L = [1], 'a'
current all partitions(eq.classes) repr:
{ 0  1  6  8 }
current all partitions:
{ 0  2  3  4  5 }
{ 1 }
{ 6  7 }
{ 8 }

=== for each [p], (split [p] w.r.t (index of L)[1],'a')
==split[0] w.r.t (index of L)[1],'a')
before split, partitions:
{ 0  2  3  4  5 }
{ 1 }
{ 6  7 }
```

```
{ 8 }

new split of [0] is [2]
after split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

before L:
L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 1 0 2
p and r are the new representatives. Now update L with the smallest of
    [0],[2]
using [p] = [0],L[r]=L[p]; L[p]=C.size();
affter L:
L:
0 1 2 3 4 5 6 7 8
2 0 0 0 0 0 1 0 2
===split[1] w.r.t (index of L)[1],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [1] is [-1]
===split[6] w.r.t (index of L)[1],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }
```

```
new split of [6] is [-1]
===split[8] w.r.t (index of L)[1],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [8] is [-1]
--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
2 0 0 0 0 0 1 0 2
Pick one [q] in L, Processing [q]= index of L = [0], 'b'
current all partitions(eq.classes) repr:
{ 0  1  2  6  8 }
current all partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

=== for each [p], (split [p] w.r.t (index of L)[0],'b')
===split[0] w.r.t (index of L)[0],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [0] is [-1]
===split[1] w.r.t (index of L)[0],'b')
before split, partitions:
StateEqRel
{ 0 }
```

```
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [1] is [-1]
==split[2] w.r.t (index of L)[0],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [2] is [-1]
==split[6] w.r.t (index of L)[0],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [6] is [-1]
==split[8] w.r.t (index of L)[0],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [8] is [-1]
--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
1 0 0 0 0 0 1 0 2
Pick one [q] in L, Processing [q]= index of L = [0], 'a'
```

```
current all partitions(eq.classes) repr:
{ 0 1 2 6 8 }
current all partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 3 4 5 }
{ 6 7 }
{ 8 }

== for each [p], (split [p] w.r.t (index of L)[0],'a')
==split[0] w.r.t (index of L)[0],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 3 4 5 }
{ 6 7 }
{ 8 }

new split of [0] is [-1]
==split[1] w.r.t (index of L)[0],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 3 4 5 }
{ 6 7 }
{ 8 }

new split of [1] is [-1]
==split[2] w.r.t (index of L)[0],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 3 4 5 }
{ 6 7 }
{ 8 }

new split of [2] is [-1]
```

```
===split[6] w.r.t (index of L)[0],'a')
 before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [6] is [-1]
===split[8] w.r.t (index of L)[0],'a')
 before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [8] is [-1]
--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 1 0 2
Pick one [q] in L, Processing [q]= index of L = [6], 'a'
current all partitions(eq.classes) repr:
{ 0  1  2  6  8 }
current all partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

=== for each [p], (split [p] w.r.t (index of L)[6],'a')
===split[0] w.r.t (index of L)[6],'a')
 before split, partitions:
StateEqRel
{ 0 }
{ 1 }
```

```
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [0] is [-1]
===split[1] w.r.t (index of L)[6],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [1] is [-1]
===split[2] w.r.t (index of L)[6],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3  4  5 }
{ 6  7 }
{ 8 }

new split of [2] is [4]
after split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

before L:
L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0 2
p and r are the new representatives. Now update L with the smallest of
    [2],[4]
using [p] = [2],L[r]=L[p]; L[p]=C.size();
```

```
affter L:
L:
0 1 2 3 4 5 6 7 8
0 0 2 0 0 0 0 0 2
===split[6] w.r.t (index of L)[6],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [6] is [-1]
===split[8] w.r.t (index of L)[6],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [8] is [-1]
--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
0 0 2 0 0 0 0 0 2
Pick one [q] in L, Processing [q]= index of L = [2], 'b'
current all partitions(eq.classes) repr:
{ 0  1  2  4  6  8 }
current all partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }
```

```
=== for each [p], (split [p] w.r.t (index of L)[2],'b')
===split[0] w.r.t (index of L)[2],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [0] is [-1]
===split[1] w.r.t (index of L)[2],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [1] is [-1]
===split[2] w.r.t (index of L)[2],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [2] is [-1]
===split[4] w.r.t (index of L)[2],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
```

```
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [4] is [-1]
===split[6] w.r.t (index of L)[2],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [6] is [-1]
===split[8] w.r.t (index of L)[2],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [8] is [-1]
--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
0 0 1 0 0 0 0 0 2
Pick one [q] in L, Processing [q]= index of L = [2], 'a'
current all partitions(eq.classes) repr:
{ 0  1  2  4  6  8 }
current all partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
```

```
{ 8 }

=== for each [p], (split [p] w.r.t (index of L)[2],'a')
===split[0] w.r.t (index of L)[2],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [0] is [-1]
===split[1] w.r.t (index of L)[2],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [1] is [-1]
===split[2] w.r.t (index of L)[2],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2  3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [2] is [3]
after split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
```

```
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

before L:
L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0 2
p and r are the new representatives. Now update L with the smallest of
   [2],[3]
using [p] = [2],L[r]=L[p]; L[p]=C.size();
affter L:
L:
0 1 2 3 4 5 6 7 8
0 0 2 0 0 0 0 0 2
胡
===split[4] w.r.t (index of L)[2],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [4] is [-1]
===split[6] w.r.t (index of L)[2],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [6] is [-1]
```

```
===split[8] w.r.t (index of L)[2],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [8] is [-1]
--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
0 0 1 0 0 0 0 0 2
Pick one [q] in L, Processing [q]= index of L = [2], 'a'
current all partitions(eq.classes) repr:
{ 0  1  2  3  4  6  8 }
current all partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

=== for each [p], (split [p] w.r.t (index of L)[2],'a')
===split[0] w.r.t (index of L)[2],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }
```

```
new split of [0] is [-1]
===split[1] w.r.t (index of L)[2],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [1] is [-1]
===split[2] w.r.t (index of L)[2],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [2] is [-1]
===split[3] w.r.t (index of L)[2],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [3] is [-1]
===split[4] w.r.t (index of L)[2],'a')
before split, partitions:
StateEqRel
{ 0 }
```

```
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [4] is [-1]
===split[6] w.r.t (index of L)[2],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [6] is [-1]
===split[8] w.r.t (index of L)[2],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [8] is [-1]
--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0 2
Pick one [q] in L, Processing [q]= index of L = [8], 'b'
current all partitions(eq.classes) repr:
{ 0  1  2  3  4  6  8 }
current all partitions:
StateEqRel
```

```
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

=== for each [p], (split [p] w.r.t (index of L)[8],'b')
===split[0] w.r.t (index of L)[8],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [0] is [-1]
===split[1] w.r.t (index of L)[8],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [1] is [-1]
===split[2] w.r.t (index of L)[8],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
```

```
{ 6  7 }
{ 8 }

new split of [2] is [-1]
===split[3] w.r.t (index of L)[8],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [3] is [-1]
===split[4] w.r.t (index of L)[8],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [4] is [-1]
===split[6] w.r.t (index of L)[8],'b')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [6] is [-1]
===split[8] w.r.t (index of L)[8],'b')
```

```
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [8] is [-1]
--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0 1
Pick one [q] in L, Processing [q]= index of L = [8], 'a'
current all partitions(eq.classes) repr:
{ 0  1  2  3  4  6  8 }
current all partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

=== for each [p], (split [p] w.r.t (index of L)[8],'a')
===split[0] w.r.t (index of L)[8],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [0] is [-1]
```

```
===split[1] w.r.t (index of L)[8],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [1] is [-1]
===split[2] w.r.t (index of L)[8],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [2] is [-1]
===split[3] w.r.t (index of L)[8],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [3] is [-1]
===split[4] w.r.t (index of L)[8],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
```

```
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [4] is [-1]
===split[6] w.r.t (index of L)[8],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [6] is [-1]
===split[8] w.r.t (index of L)[8],'a')
before split, partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4  5 }
{ 6  7 }
{ 8 }

new split of [8] is [-1]
--- while each [q], (split [p] w.r.t ([index of L]=[q],a))
L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0 0
```

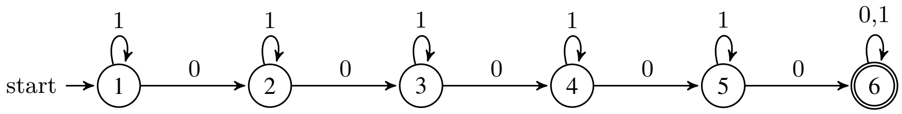


图 1.2: Minimizing example

{a,b},{d,e}is not equivalent states.
Sets of equivalent states: {a,c},{b},{d},{e}

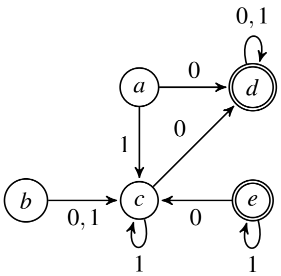


图 1.3: Finite state automaton

1.1 Minimization by equivalence of states

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.

Hopcroft71. Hopcroft, J.E. *An n log n algorithm for minimizing states in a finite automaton*, in The Theory of Machines and Computations (Z. Kohavi, ed.), pp.180-196, Academic Press, New York, 1971.

Gries73. Gries, D. *Describing an Algorithm by Hopcroft*, Acta Inf. 2:97 109, 173. © by Springer-Verlag 1973

Knuutila2001. Knuutila, T. *Re-describing an Algorithm by Hopcroft.* Theoret. Computer Science 250 (2001) 333–363.

Ratnesh95. Ratnesh Kumar, *Modeling and Control of Logical Discrete Event Systems*, © 1995 by Springer Science+Business Media New York.

Jean2011. Jean Berstel, Luc Boasson, Olivier Carton, Isabelle Fagnot , *Minimization of automata*, Universit*é* Paris-Est Marne-la-Vall*é*e 2010 Mathematics Subject Classification: 68Q45, 2011.

Kenneth2012. Kenneth H. Rosen 著, 徐六通译, 离散数学及其应用 *Discrete Mathematics and Its Applications*,seventh Edition, 2012, 机械工业出版社, 北京, 2014.