Autometa

<<Autometa2.ppt>>

- Regular expressions are defined as follows. Each regular expression denotes a set of strings (a regular set).
  - ψ is a regular expression denoting the empty set (the set containing no strings).
  - λ is a regular expression denoting the set that contains only the empty string. Note that this set is not the same as the empty set, because it contains one element.

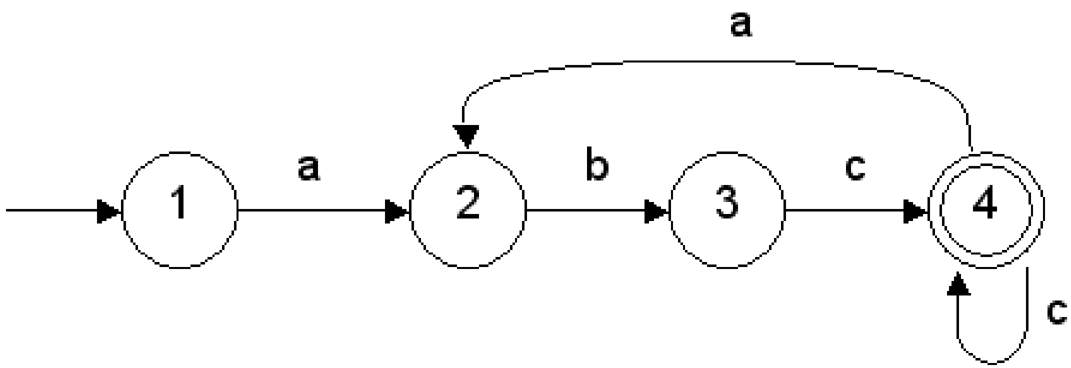
- A string S is a regular expression denoting a set containing only S. If S contains meta-characters, S can be quoted to avoid ambiguity.
- If A and B are regular expressions, then A|B, AB, A* are also regular expressions, denoting the alternation, catenation, and Kleene closure of the corresponding regular sets.

- Any finite set of strings can be represented by a regular expression of the form $(s_1|s_2|\ldots s_k)$.
- We often utilize the following operations as a notational convenience. They are not strictly necessary, because their effect can be obtained (albeit somewhat clumsily) using the three standard regular operators (alternation, catenation, Kleene closure) :

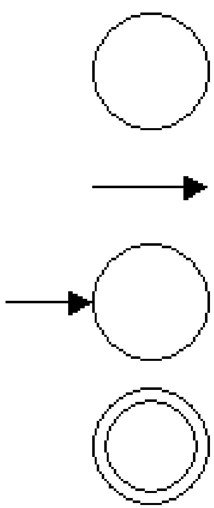
- P^+ denotes all strings consisting of one or more strings in P catenated together : $P^* = (P^+|\lambda)$ and $P^+ = PP^*$.
- If A is a set of characters, Not(A) denotes (V - A); that is, all characters in V not included in A. Since Not(A) is finite, it is trivially regular. It is possible to extend Not to strings, rather than just V. That is, if S is a set of strings, we can define Not(S) to be (V^* - S). Although it may be infinite, this set is also regular (see Exercise 20).

- Finite automaton (FA) can be used to recognize the tokens specified by a regular expression. An FA is a simple, idealized computer that recognizes strings belonging to regular sets. It consists of :
  - A finite set of states
  - A set of transitions (or moves) from one state to another, labeled with characters in V
  - A special start state
  - A set of final, or accepting, states

- Finite automata can be represented graphically using transition diagrams :



L = abc(c|abc)*



is a state

is a translation

is the start state

is a final state

Transition table

State	a	b	c
1	2		
2		3	
3			4
4	1		4

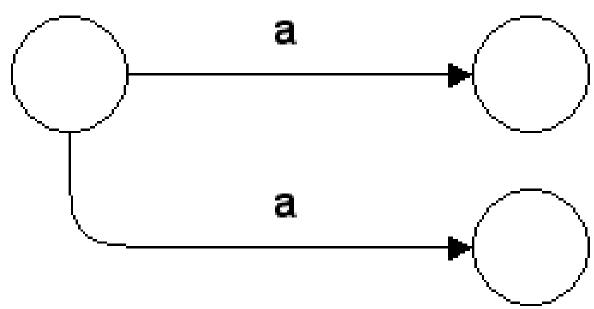


Figure 3.7 An NFA with Two a Transitions

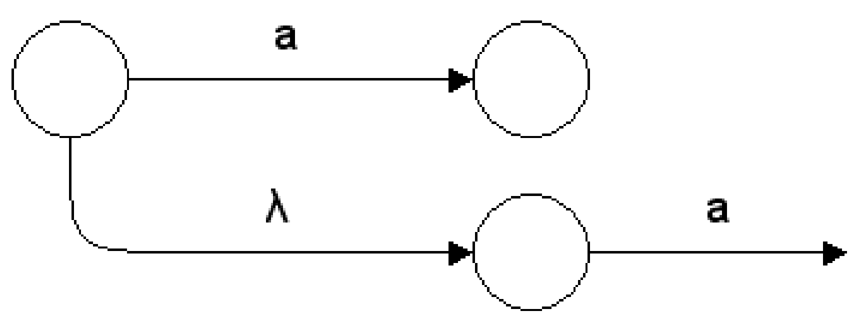


Figure 3.8 An NFA with a λ Transitions

- The algorithm to make an FA from a regular expression proceeds in two steps : First, it transforms the regular expression into an NFA, and then it transforms the NFA into a deterministic one. This first step is very easy. In fact, we can transform any regular expression into an NFA with the following properties :
  - There is a unique final state.
  - The final state has no successors.
  - Every other state has rather one or two successors.

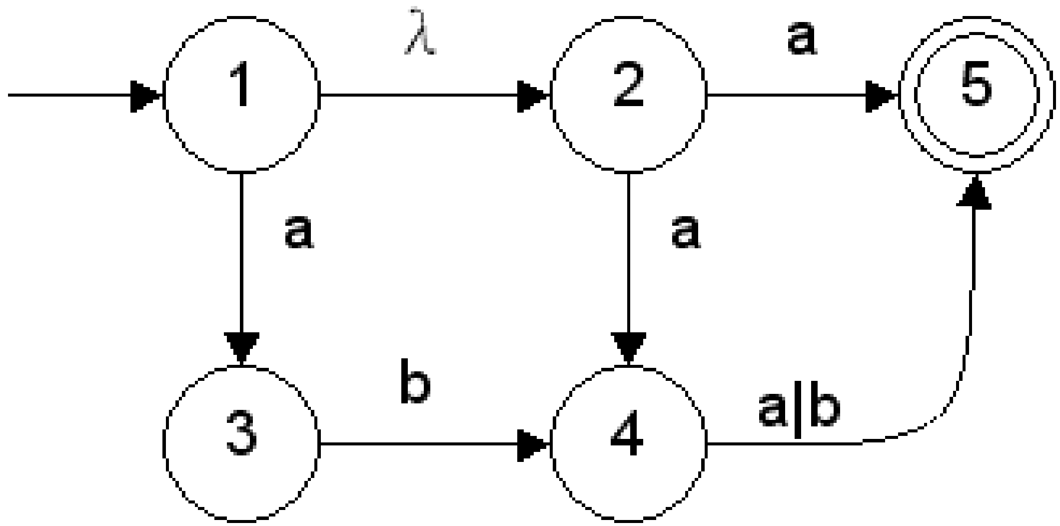
```
/ *
*Add to S all states reachable from it
*Using only λ transitions of N
*/
void close ( set_of_fa_states *S)
{
   while ( there is a state x in S and a state y not in S such that x→y using a λ transition )
      add y to S
}
```

using this procedure, we can define the construction of M :

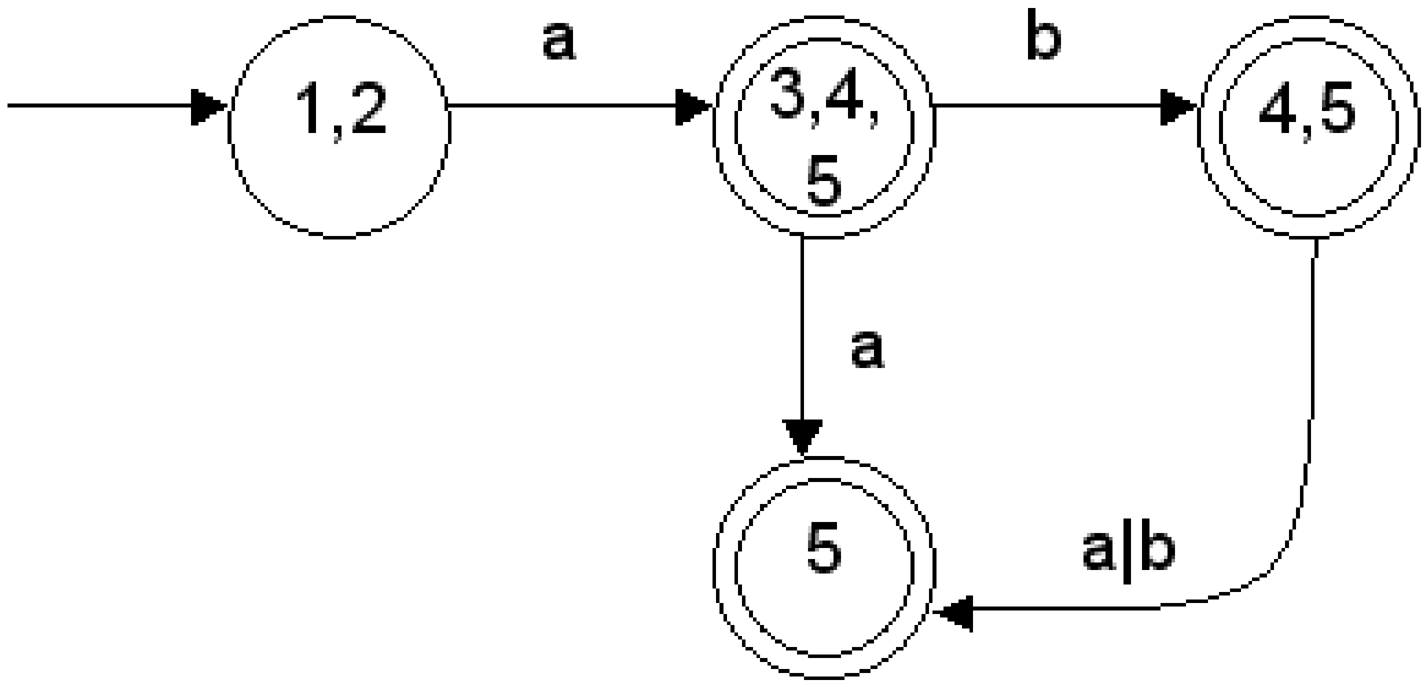
```
 void make_deterministic ( nondeterministic_fa N, deterministic_fa *M )
{
//next page…
```

```
set_of_fa_states T;
M->initial_state = SET_OF(N.initial_state);
Close(& M->initial_state);
Add M->initial_state to M->states;
While(states or transitions can be added)
{
    choose S in M->states and c in Alphabet;
    T = SET_OF(y in N.states
               SUCH_THAT x -c-> y for some x in S);
    close(& T);
    if(T not in M->states)
        add T to M->states;
    Add the transition to M->transitions: S -c-> T;
}
M->final_states =
SET_OF(S in M->states SUCH_THAT N.final_state in S);
}
```

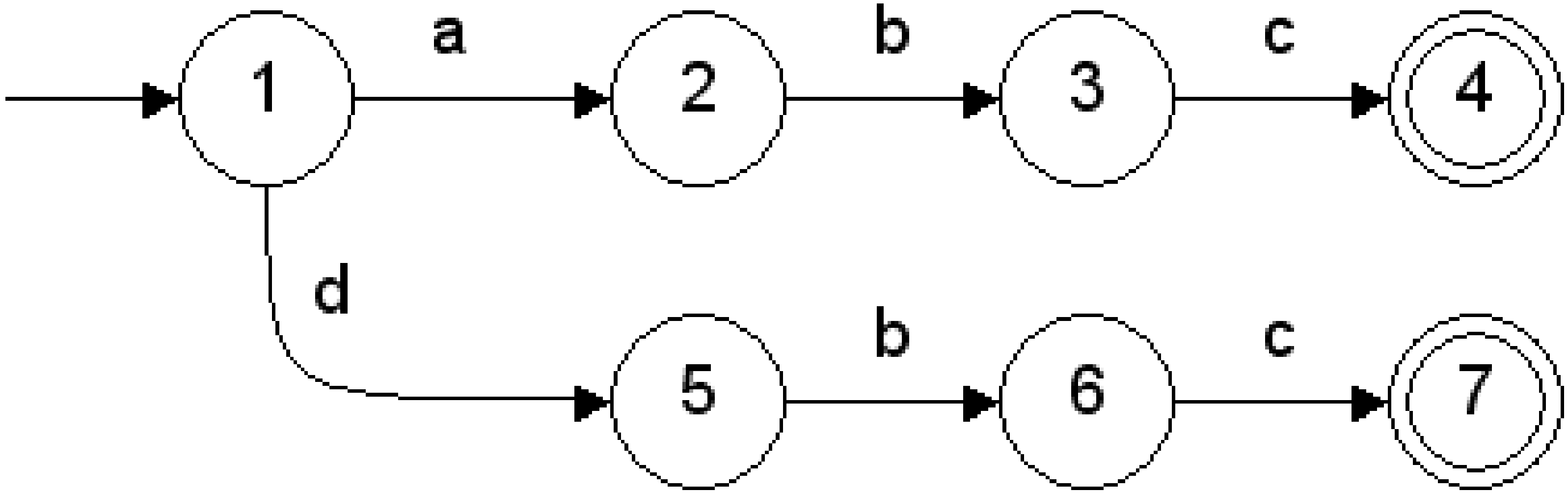
NFA :



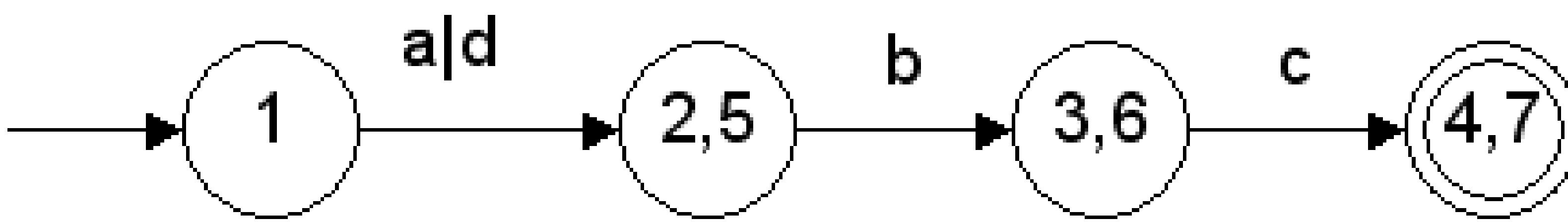
We start with state 1, the start state of N, and add state 2, its λ-successor. Hence M's start state is {1,2}. Neither state 1 nor state 2 has a successor under b. Under a, {1,2}'s successor is {3,4,5}. {3,4,5}'s successors under a and b are {5} and {4,5}. {4,5}'s successor under b is {5}. Final states of M are those state sets that contain N's final state (5). The resulting DFA is :



State	a	b
{1,2}	{ 3,4,5 }	∅
{ 3,4,5 }	{ 5 }	{ 4,5 }
{ 5 }	∅	∅
{ 4,5 }	{ 5 }	{ 5 }



減少 DFA state 數！



initial state

{ 1,2,3,5,6 } {4,7} final

$\overline{12356}$ $\overline{47}$

$\overline{125}$ $\overline{36}$ $\overline{47}$

$\overline{1}$ $\overline{25}$ $\overline{36}$ $\overline{47}$

State	a	b	c	d
1	2	$\varnothing$	$\varnothing$	5
2	$\varnothing$	3	$\varnothing$	$\varnothing$
3	$\varnothing$	$\varnothing$	4*	$\varnothing$
5	$\varnothing$	6	$\varnothing$	$\varnothing$
6	$\varnothing$	$\varnothing$	7*	$\varnothing$

merge

```
void split ( set_of_fa_states *ss )
{
    do {
        Let S be any merged state corresponding to
        {s1, … ,sn} and let c be any character;
        Let t1, … ,tn be the successor states to
        {s1, … ,sn}
            under c;
        if(t1, … ,tn do not all belong to the same
            merged state)
        {
            Split S into new states so that si and sj
            remain in the same merged state if and
            only if ti and tj are in the same merged
            state;
        }
    }while(more splits are possible);
}
```