CS 4300: Compiler Theory

Chapter 3
Lexical Analysis

Dr. Xuejun Liang

Outlines (Sections)

1. The Role of the Lexical Analyzer
2. Input Buffering (Omit)
3. Specification of Tokens
4. Recognition of Tokens
5. The Lexical -Analyzer Generator Lex
6. Finite Automata
7. From Regular Expressions to Automata
8. Design of a Lexical-Analyzer Generator
9. Optimization of DFA-Based Pattern Matchers*

Quick Review of Last Lecture

From Regular Expressions to Automata

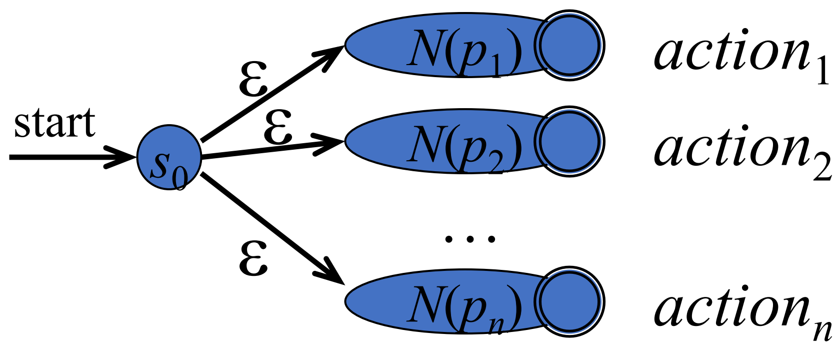
- Conversion of an NFA into a DFA
  - ***Subset construction*** algorithm
    - Initial ε*-closure, then move and* ε*-closure*
  - Subset Construction Examples
  - Simulating an NFA Using ε*-closure* and *move*
- From Regular Expression to NFA
  - **Thompson's Construction** algorithm
  - Thompson's Construction Example

8. Design of a Lexical-Analyzer Generator
Construct an NFA from a Lex Program

Lex specification with regular expressions

p_1	$\{$ *action*$_1$ $\}$
p_2	$\{$ *action*$_2$ $\}$
…	
p_n	$\{$ *action*$_n$ $\}$

NFA

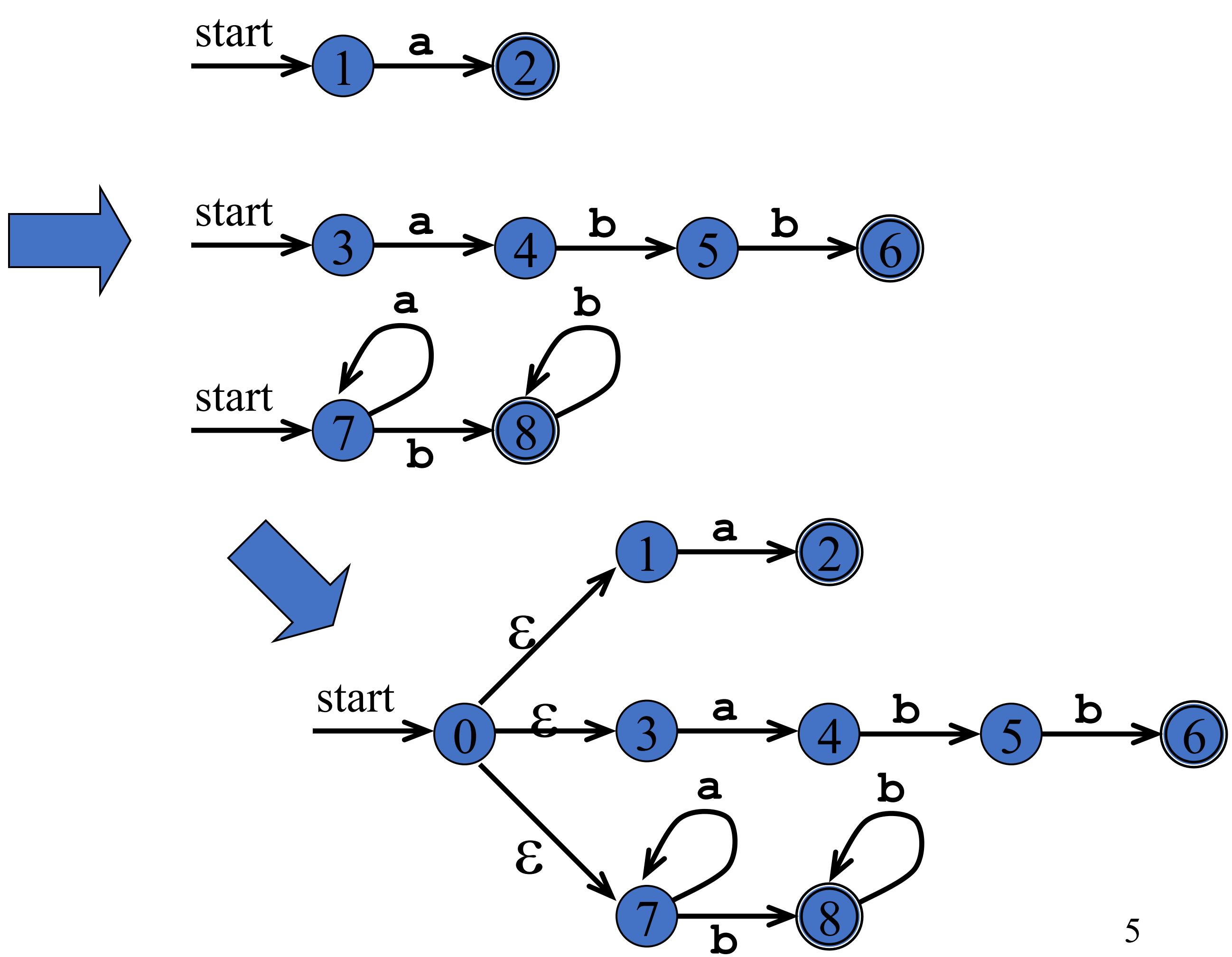


Subset construction

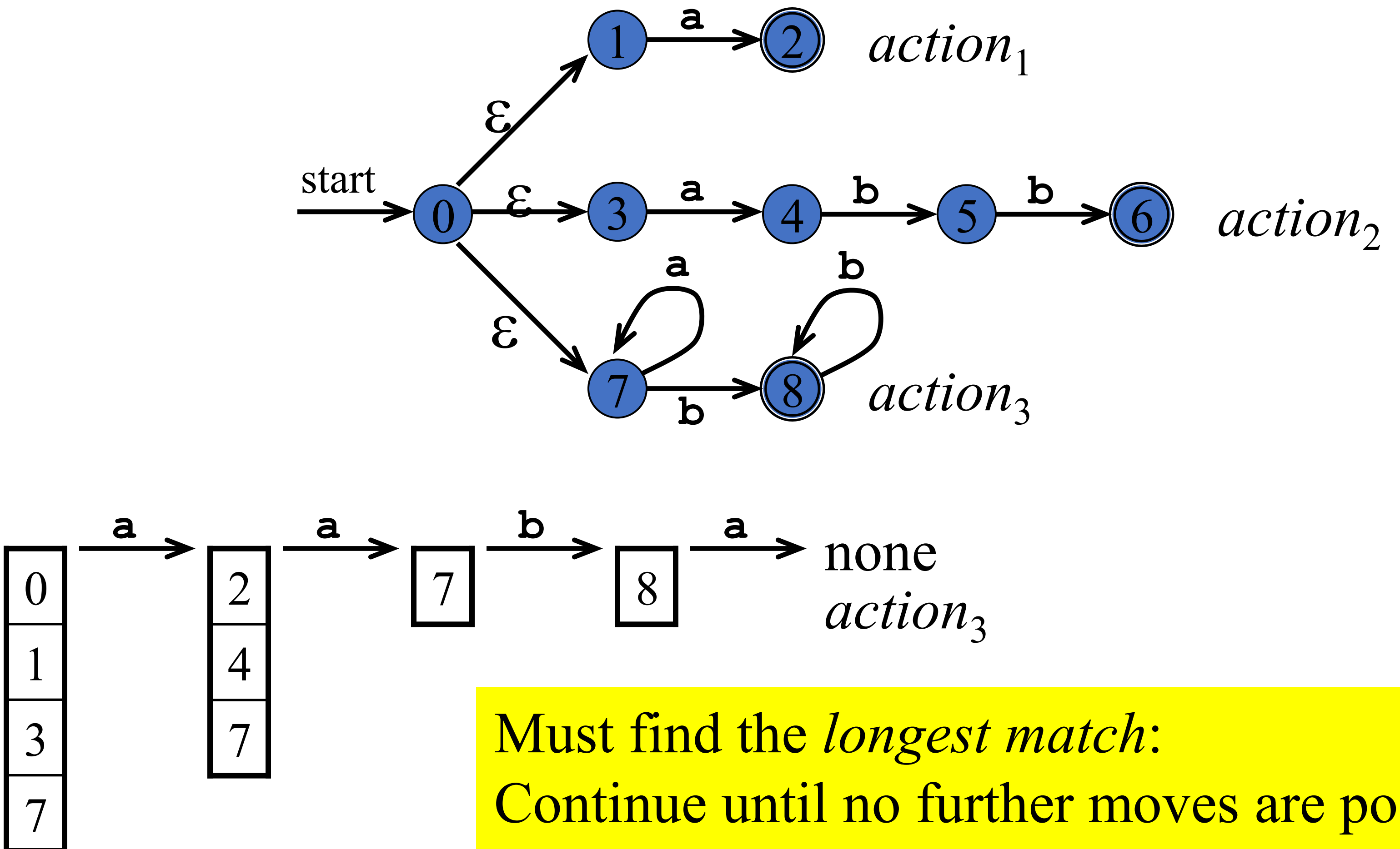
DFA

Combining the NFAs of a Set of Regular Expressions

a { *action*$_1$ }
abb { *action*$_2$ }
a*b+ { *action*$_3$ }

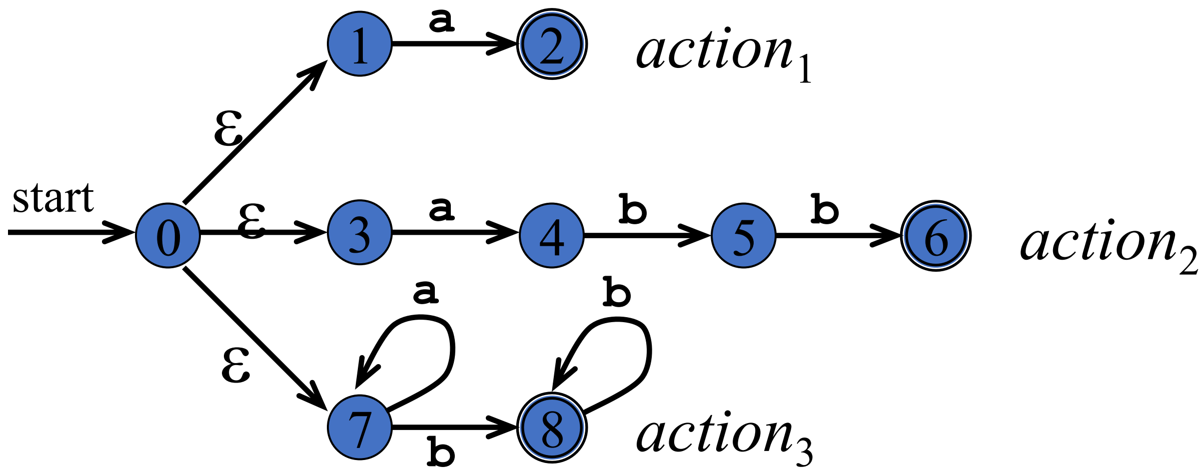


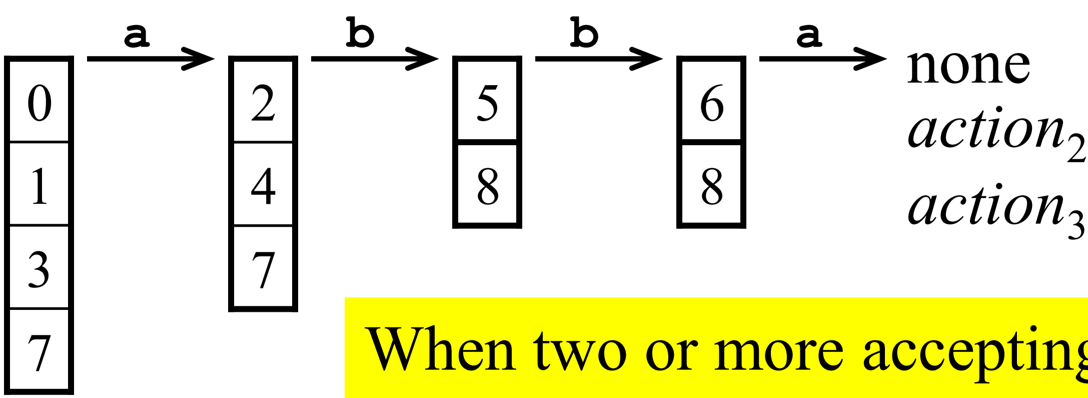
Simulating the Combined NFA Example 1



Must find the *longest match*:
Continue until no further moves are possible
When last state is accepting: execute action

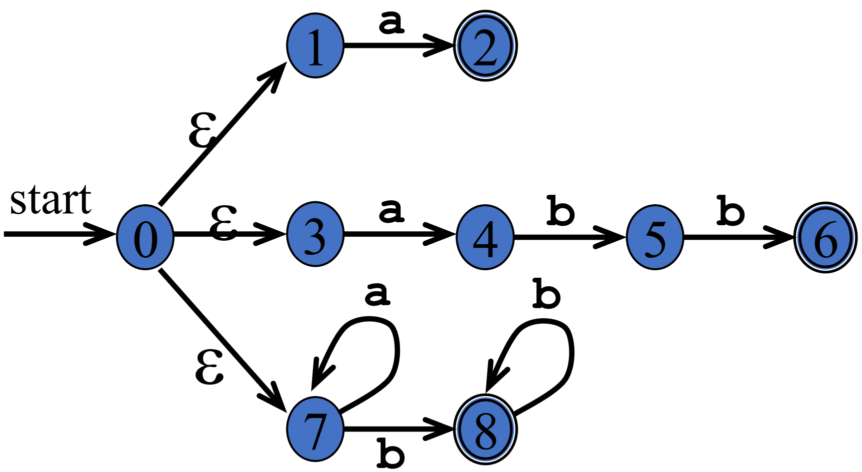
Simulating the Combined NFA Example 2





When two or more accepting states are reached, the first action given in the Lex specification is executed

DFA's for Lexical Analyzers

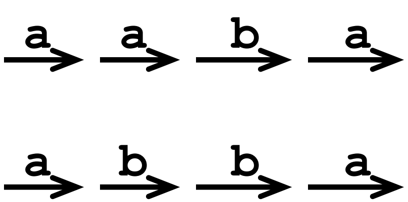


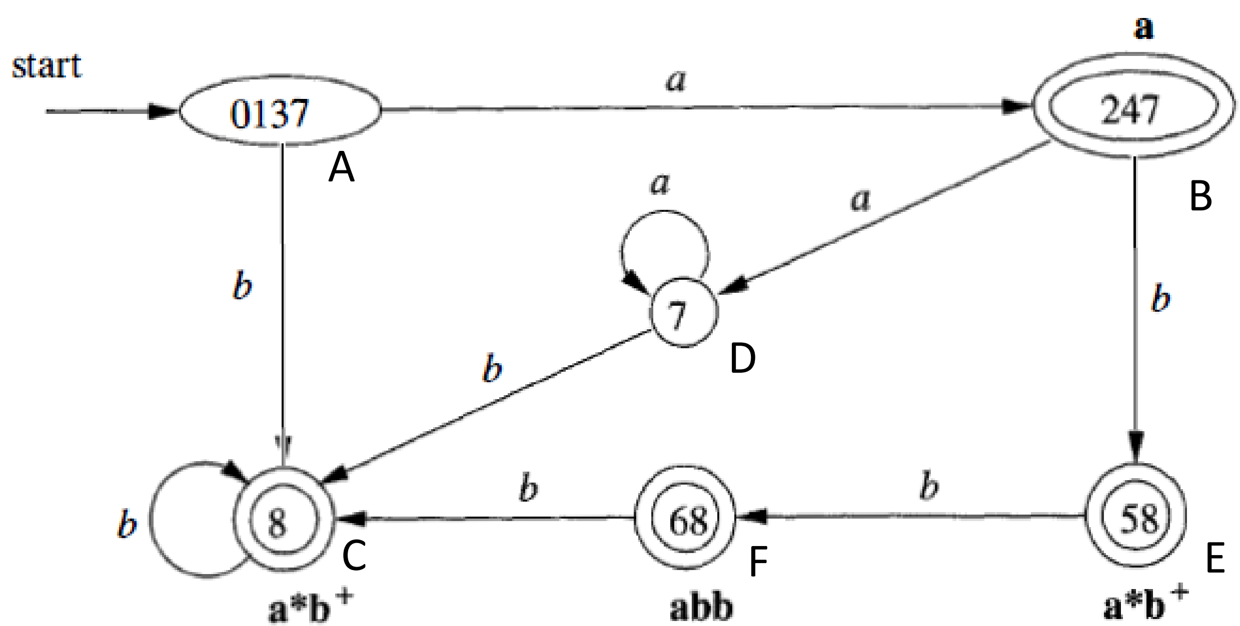
NFA

Subset construction

DFA

Examples

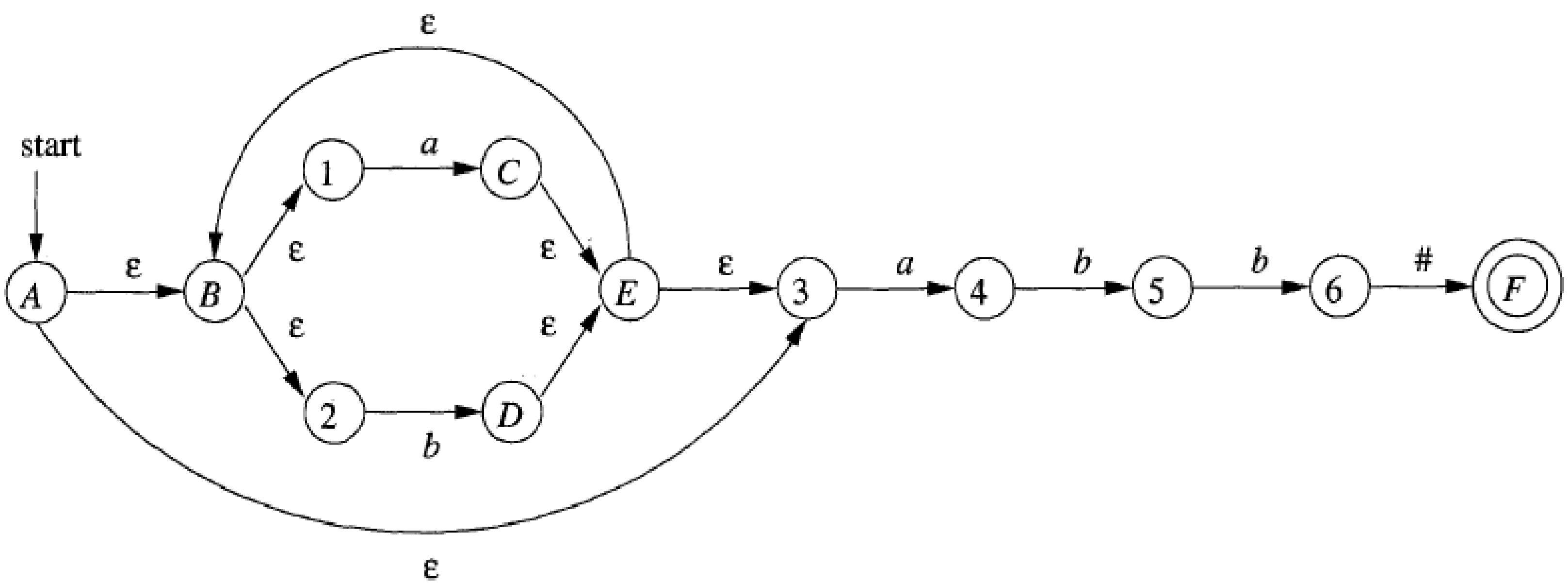




9. From RE to DFA Directly

- The "*important states*" of an NFA are those without an ε-transition, that is if *move*({*s*},*a*) ≠ ∅ for some *a* then *s* is an important state
- The subset construction algorithm uses only the important states when it determines ε-*closure*(*move*(*T*,*a*))

NFA Constructed for (**a**|**b**)***abb#**



Note:

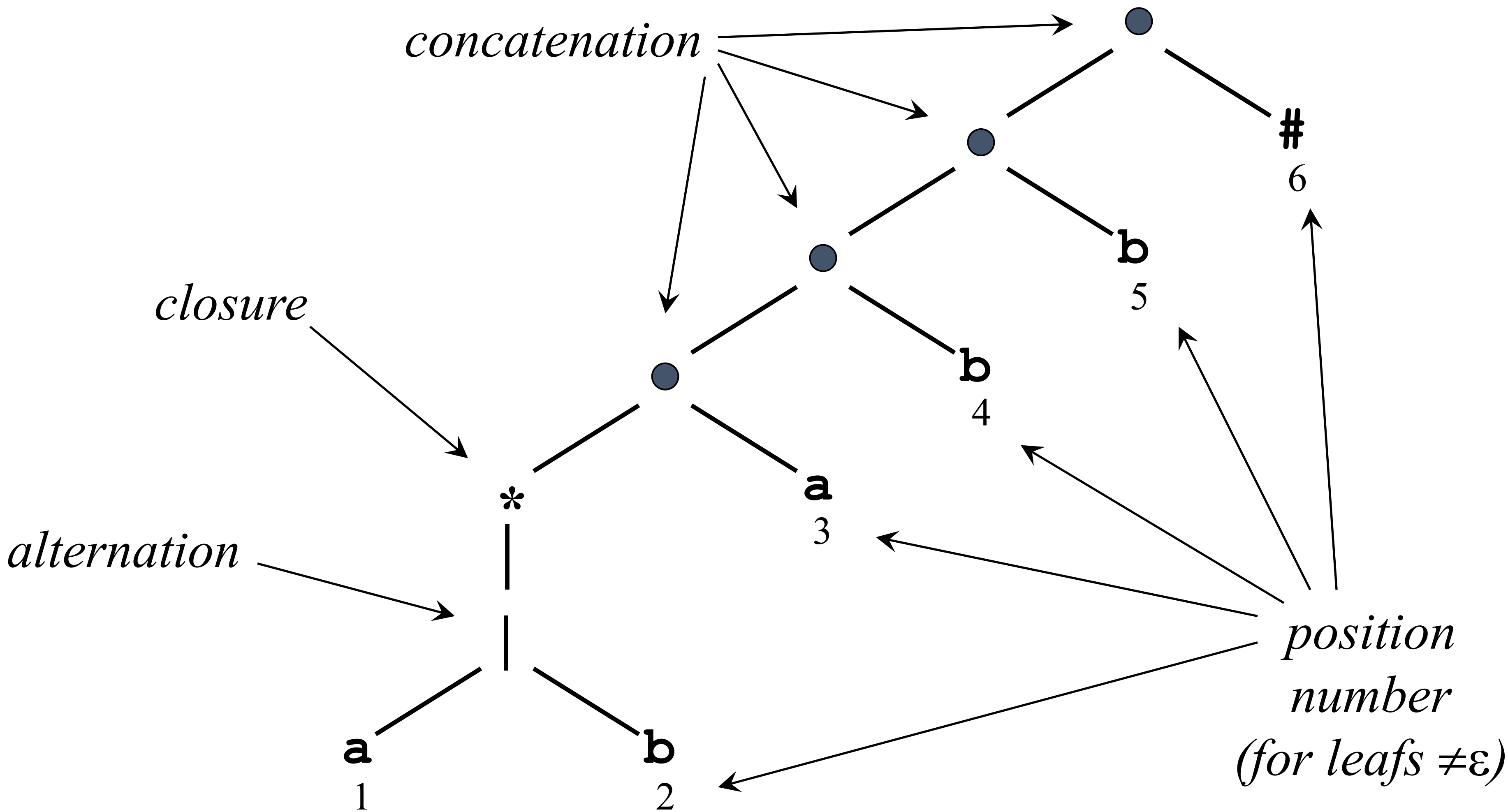
1. The NFA is constructed by Thompson's Algorithm
2. The important states in the NFA are numbered

Algorithm:

INPUT : A regular expression r.
OUTPUT: A DFA D that recognizes L(r) .

1. Augment the regular expression *r* with a special end symbol # to make accepting states important: the new expression is *r#*
2. Construct a syntax tree T from *r#*
3. Traverse the tree to construct functions *nullable*, *firstpos*, *lastpos*, and *followpos*
4. Construct *Dstates*, the set of states of DFA D, and *Dtran*, the transition function for D.
5. The start state of D is *firstpos*(n_0), where node n_0 is the root of T. The accepting states are those containing the position for the end marker symbol #.

Syntax Tree of (**a**|**b**)***abb#**



Functions Computed from the Syntax Tree

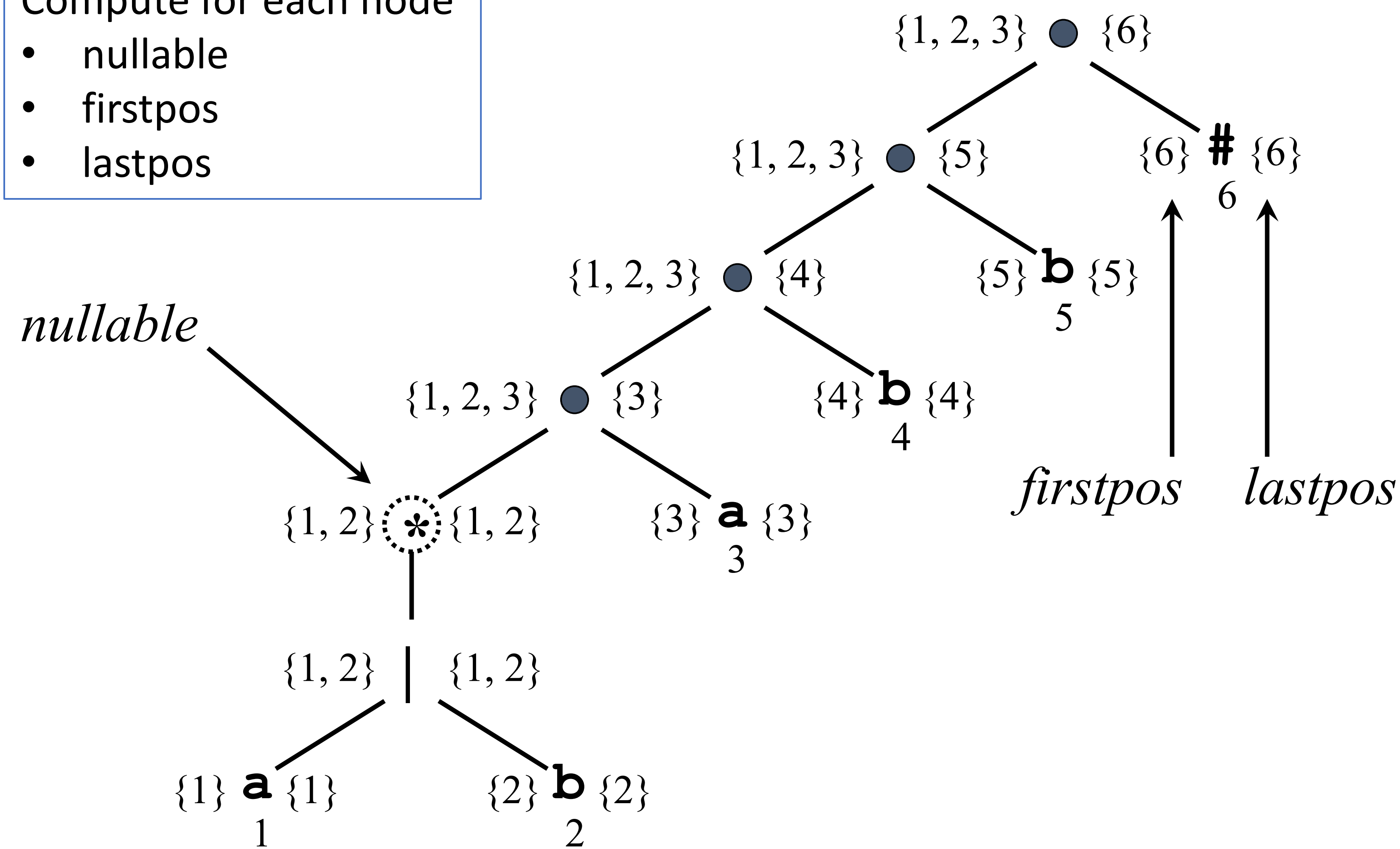
- *nullable*(*n*): is true for a syntax-tree node *n* if and only if the subexpression represented by *n* has ε in its language.
- *firstpos*(*n*): set of positions that can match the first symbol of a string generated by the subexpression represented by node *n*
- *lastpos*(*n*): the set of positions that can match the last symbol of a string generated by the subexpression represented by node *n*
- *followpos*(*p*): the set of positions that can follow position *p* in the syntax-tree

Functions Computed from the Syntax Tree (cont.)

Node n	$nullable(n)$	$firstpos(n)$	$lastpos(n)$
Leaf ε	true	$\varnothing$	$\varnothing$
Leaf i	false	$\{i\}$	$\{i\}$
| / \ c_1 c_2	$nullable(c_1)$ or $nullable(c_2)$	$firstpos(c_1)$ $\cup$ $firstpos(c_2)$	$lastpos(c_1)$ $\cup$ $lastpos(c_2)$
• / \ c_1 c_2	$nullable(c_1)$ and $nullable(c_2)$	**if** $nullable(c_1)$ **then** $firstpos(c_1) \cup$ $firstpos(c_2)$ **else** $firstpos(c_1)$	**if** $nullable(c_2)$ **then** $lastpos(c_1) \cup$ $lastpos(c_2)$ **else** $lastpos(c_2)$
* | c_1	true	$firstpos(c_1)$	$lastpos(c_1)$

Annotated Syntax Tree of (**a**|**b**)***abb#**

Compute for each node
- nullable
- firstpos
- lastpos



Algorithm: *followpos*

Initially, all *followpos*(i) = Φ

for each node n in the tree **{**

    if n is a cat-node with left child c_1 and right child c_2

        for each i in *lastpos*(c_1) **{**

            followpos(i) := *followpos*(i) $\cup$ *firstpos*(c_2)

        }

    else if n is a star-node

        for each i in *lastpos*(n) **{**

            followpos(i) := *followpos*(i) $\cup$ *firstpos*(n)

        }

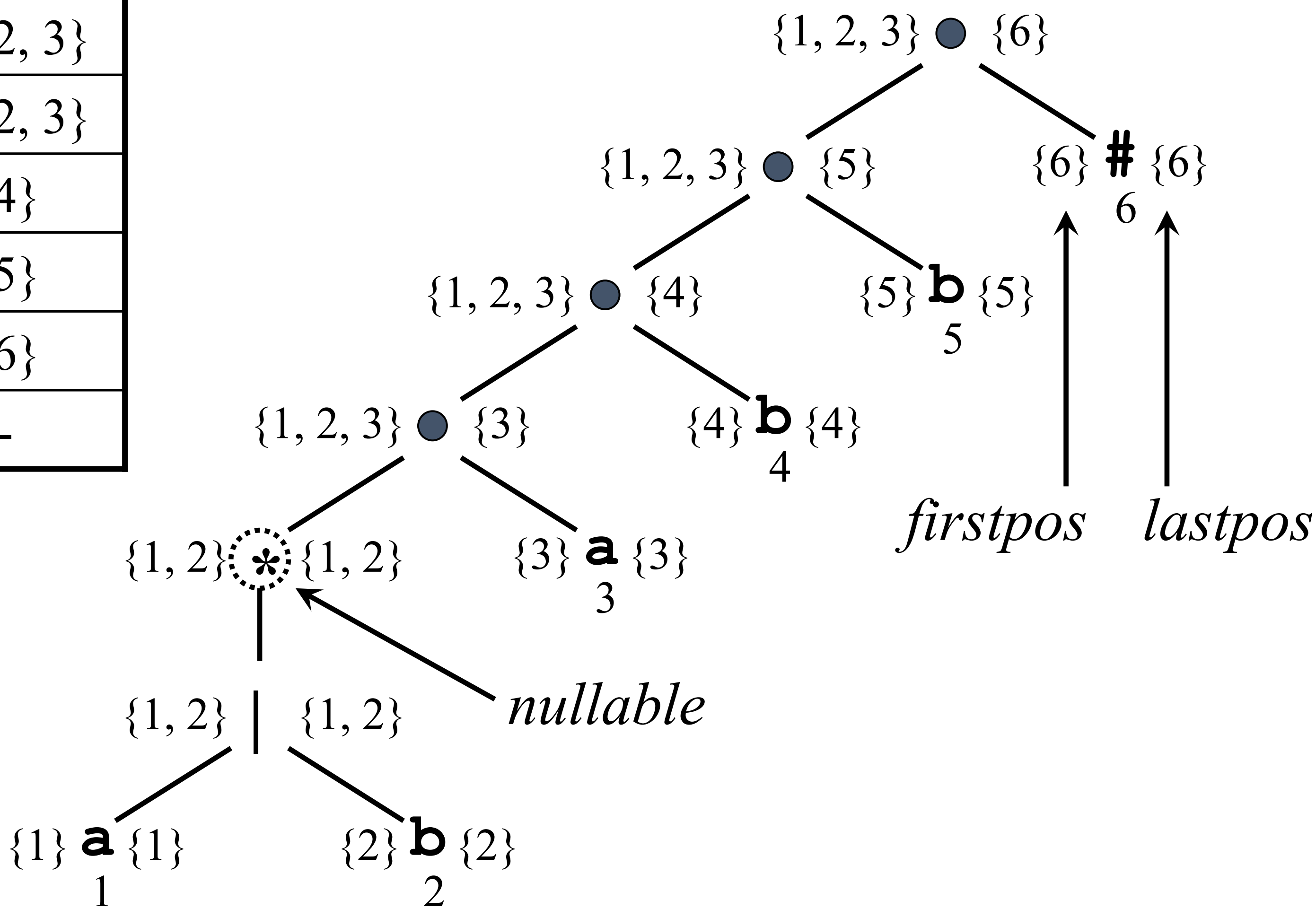
}

```
  •
 / \
c1   c2

  *
  |
  c1
```

followpos: Example

Node	*followpos*
1(a)	{1, 2, 3}
2(b)	{1, 2, 3}
3(a)	{4}
4(b)	{5}
5(b)	{6}
6(#)	-

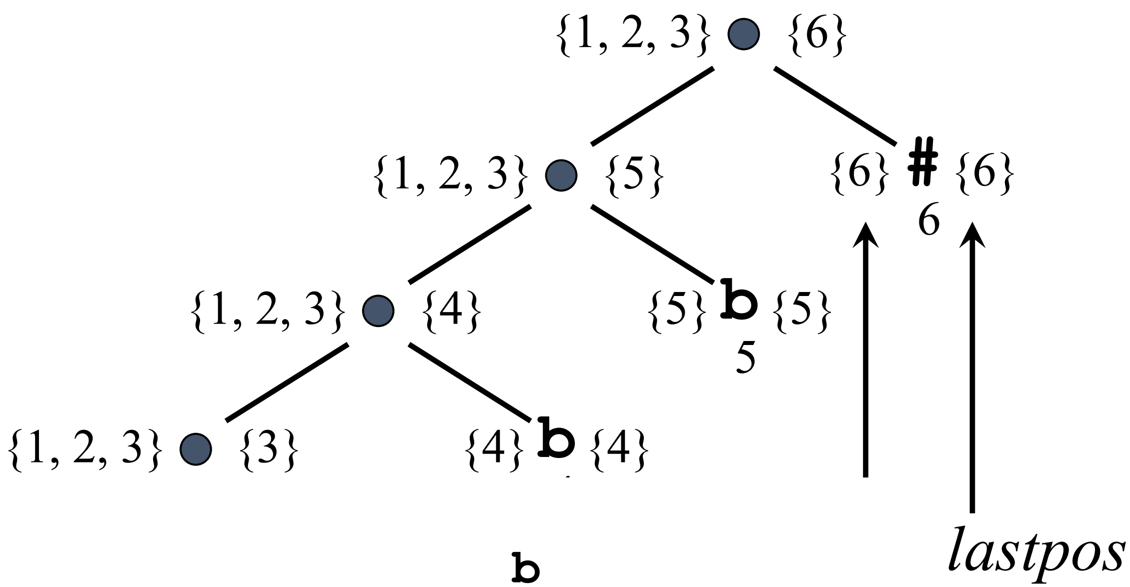


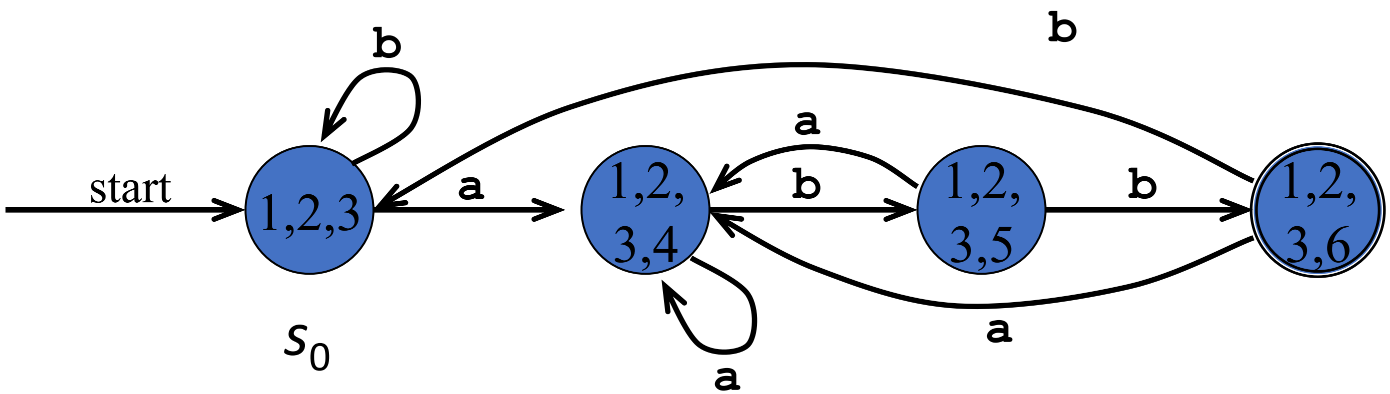
Algorithm: Construct *Dstates*, and *Dtran*

$s_0 = \mathit{firstpos}(n_0)$ where n_0 is the root of the syntax tree
Dstates := $\{s_0\}$ and s_0 is unmarked
while (there is an unmarked state *S* in Dstates) {
    mark *S;*
    for each input symbol $a \in \Sigma$ {
        let U be the union of *followpos*(p) for all p
            in S that correspond to a;
        if (*U* not in *Dstates*)
            add *U* as an unmarked state to *Dstates*
        Dtran[*S*,*a*] = *U*
    }
}

From RE to DFA Directly: Example (1)

$s_0 = firstpos(n_0)$
$= \{1,2,3\}$



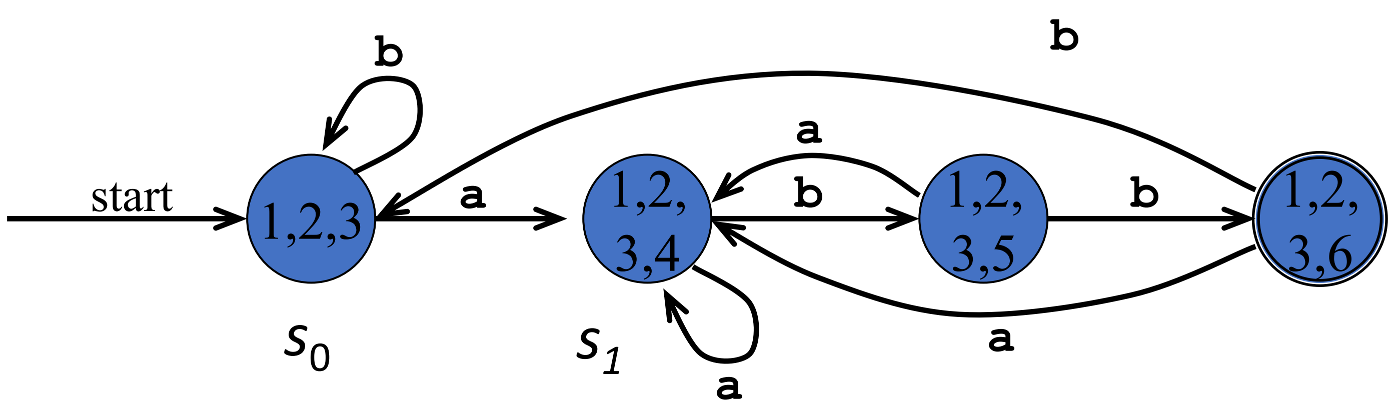


From RE to DFA Directly: Example (2)

Node	*followpos*
1(a)	{1, 2, 3}
2(b)	{1, 2, 3}
3(a)	{4}
4(b)	{5}
5(b)	{6}
6(#)	-

$Dtran[\{1,2,3\}, a]$
$= followpos(1) \cup followpos(3)$
$= \{1, 2, 3, 4\} = S_1$

$Dtran[\{1,2,3\}, b]$
$= followpos(2)$
$= \{1, 2, 3\}$

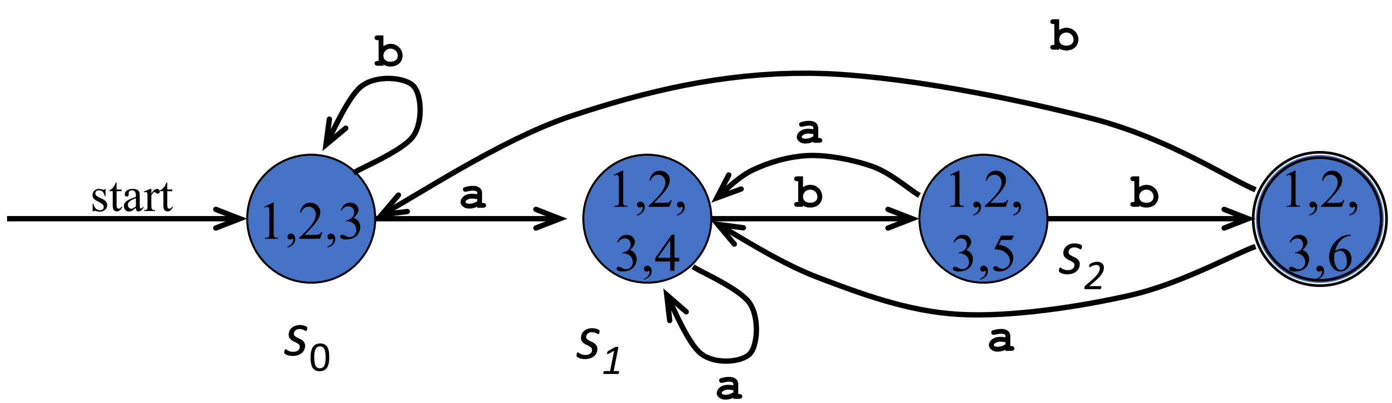


From RE to DFA Directly: Example (3)

Node	*followpos*
1(a)	{1, 2, 3}
2(b)	{1, 2, 3}
3(a)	{4}
4(b)	{5}
5(b)	{6}
6(#)	-

Dtran[{*1,2,3,4*}, *a*]
= *followpos*(1) ∪ *followpos*(3)
={1, 2, 3, 4}

Dtran[{*1,2,3,4*}, *b*]
= *followpos*(2) ∪ *followpos*(4)
={1, 2, 3, 5} = S_2

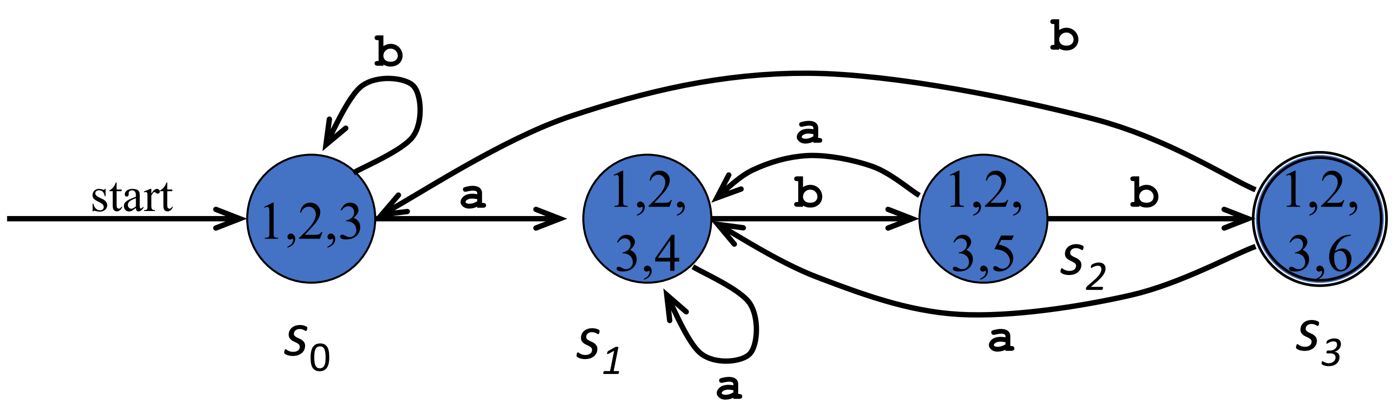


From RE to DFA Directly: Example (4)

Node	*followpos*
1(a)	{1, 2, 3}
2(b)	{1, 2, 3}
3(a)	{4}
4(b)	{5}
5(b)	{6}
6(#)	-

Dtran[{*1,2,3,5*}, *a*]
= *followpos*(1) ∪ *followpos*(3)
={1, 2, 3, 4}

Dtran[{*1,2,3,5*}, *b*]
= *followpos*(2) ∪ *followpos*(5)
={1, 2, 3, 6} = S_3



From RE to DFA Directly: Example (5)

Node	*followpos*
1(a)	{1, 2, 3}
2(b)	{1, 2, 3}
3(a)	{4}
4(b)	{5}
5(b)	{6}
6(#)	-

Dtran[{*1,2,3,6*}, *a*]
= *followpos*(1) ∪ *followpos*(3)
={1, 2, 3, 4}

Dtran[{*1,2,3,6*}, *b*]
= *followpos*(2)
={1, 2, 3}