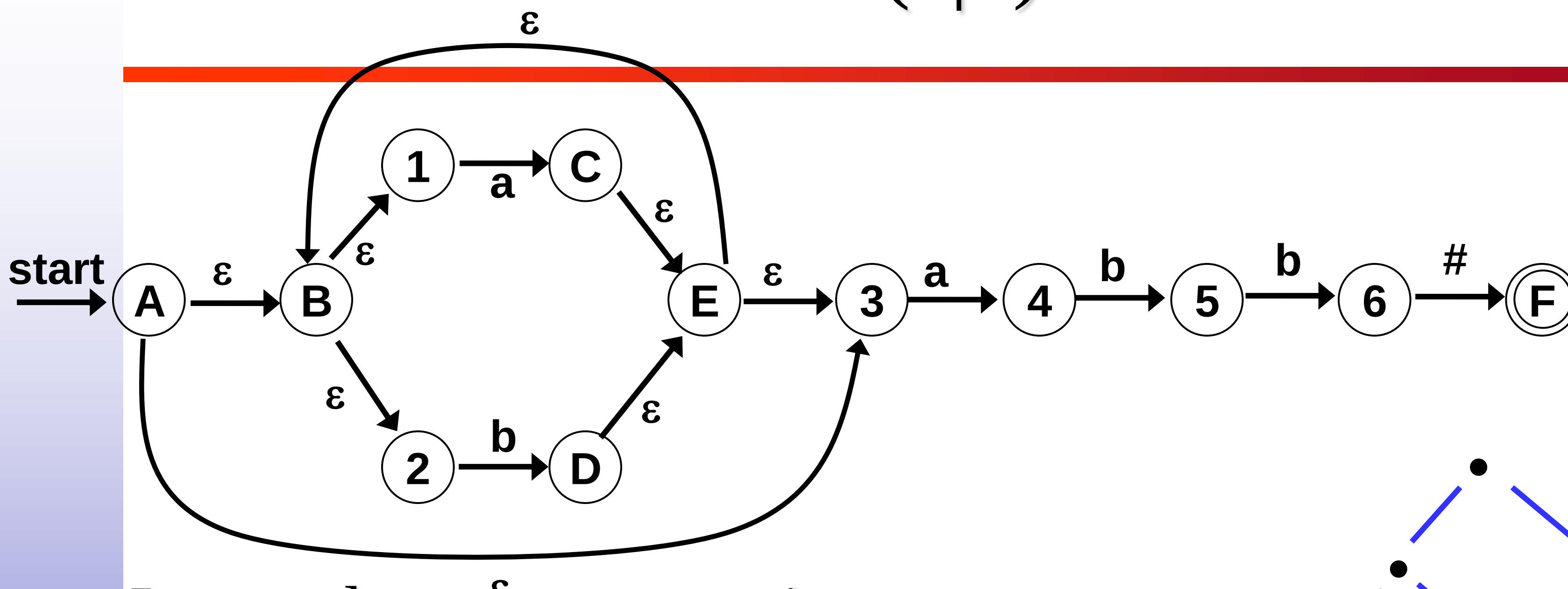
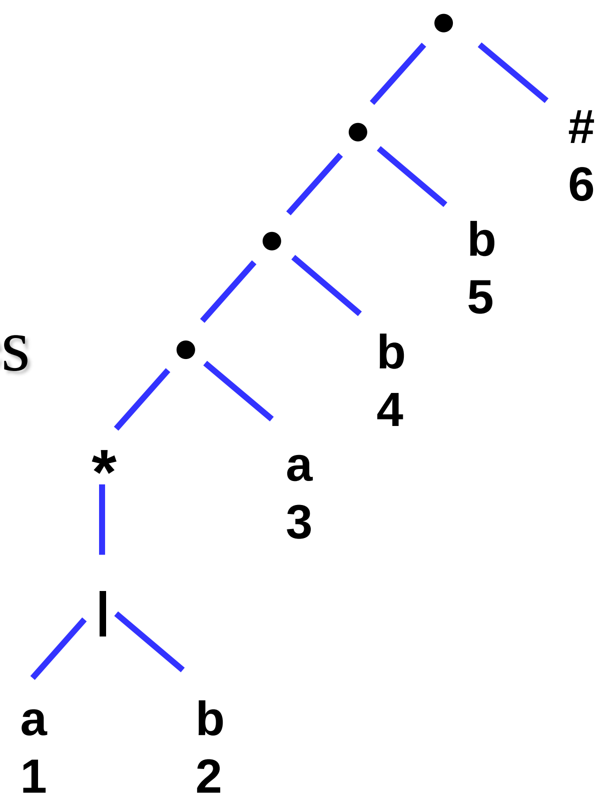
CSE420 Compiler Design

Lecture: 3
Lexical Analysis (Part 5)

NFA for (a|b)*abb#



- Lettered states are non-important states
- Number states are important states
  - Numbers correspond to the number in syntax tree



Converting Regular Expression to DFA

- We may convert a regular expression into a DFA (without creating a NFA first).
- First we augment the given regular expression by concatenating it with a special symbol #.

  r ➔ (r)#

  augmented regular expression
- Then, we create a syntax tree for this augmented regular expression.

Construction of DFA from RE

- Input: A regular expression r
- Output: A DFA D that recognizes L(r)

- Method:

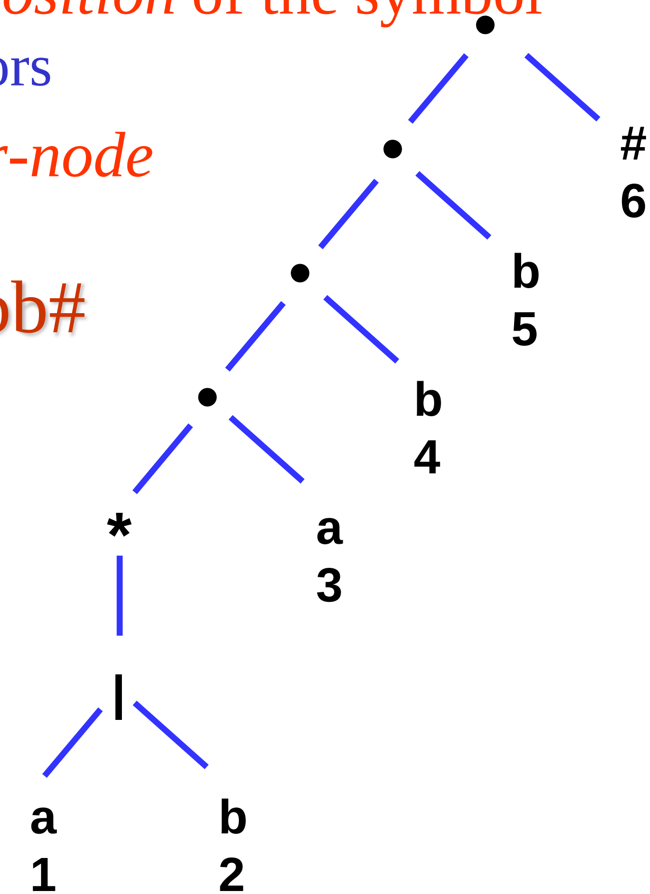
1. Construct syntax tree ST for augmented RE r#
2. Construct the functions nullable, firstpos, lastpos and followpos for ST
3. Construct Dstates: set of states of D

   Dtrans: transition table for D

Syntax Tree

- Augmented RE (r)# can be represented by a syntax tree
  - Leaves contain: Alphabet symbols or ε
    - Each non-ε leaf is associated with a unique number-*position* of the leaf and *position* of the symbol
  - Internal nodes contain: Operators
    - *cat-node*, *or-node* or *star-node*

- Syntax tree for r# = (a|b)*abb#



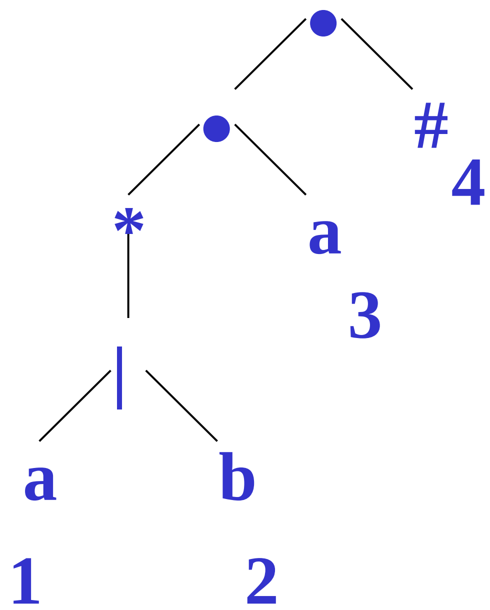
firstpos, lastpos, nullable

To evaluate followpos, we need three more functions to be defined for the nodes (not just for leaves) of the syntax tree.

- **firstpos(n) -- the set of the positions of the first symbols of strings generated by the sub-expression rooted by n.**
- **lastpos(n) -- the set of the positions of the last symbols of strings generated by the sub-expression rooted by n.**
- **nullable(n) -- *true* if the empty string is a member of strings generated by the sub-expression rooted by n *false* otherwise**

Regular Expression ➔ DFA (cont.)

$(a|b)^* a$ ➔ $(a|b)^* a$ # augmented regular expression



Syntax tree of $(a|b)^* a$

- **each symbol is numbered**
- **each symbol is at a leaf**
- **inner nodes are operators**

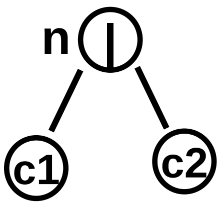
Terminology

- Nullable:
  - Nodes that are the root of some sub-expression that generate empty string

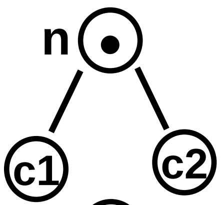
- If n is a leaf labeled by ε then
  - **nullable (n) = true**

- If n is a leaf labeled with position *i*
  - **nullable (n) = false**

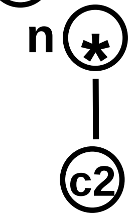
- If n is an or-node (|) with children c1 and c2
  - **nullable (n) = nullable(c1) or nullable (c2)**



- If n is an cat-node (•) with children c1 and c2
  - **nullable (n) = nullable(c1) and nullable (c2)**



- If n is an star-node (*) with children c1
  - **nullable (n) = true**



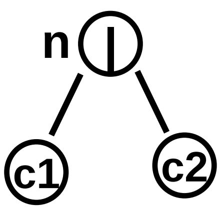
Terminology

- Firstpos(n):
  - Set of positions that can match the first symbol of a string generated by the sub-expression rooted at n

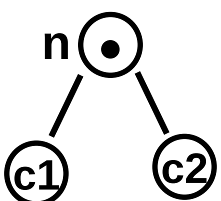
- If n is a leaf labeled by ε then
  - **firstpos (n) = $\varnothing$**

- If n is a leaf labeled with position *i*
  - **firstpos (n) = {i}**

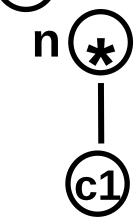
- If n is an or-node (|) with children c1 and c2
  - **firstpos (n) = firstpos(c1) $\cup$ firstpos (c2)**



- If n is a cat-node (•) with children c1 and c2
  - **firstpos(n) = If nullable (c1) then firstpos(c1) $\cup$ firstpos (c2)
else firstpos(c1)**



- If n is an star-node (*) with children c1
  - **firstpos (n) = firstpos(c1)**



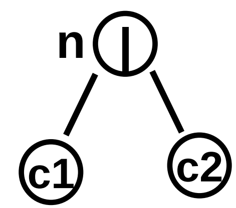
Terminology

- Lastpos(n):
  - Set of positions that can match the last symbol of a string generated by the sub-expression rooted at n

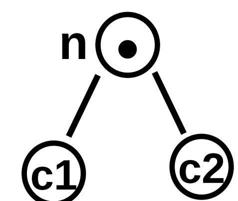
- If n is a leaf labeled by ε then
  - **lastpos (n) = $\varnothing$**

- If n is a leaf labeled with position *i*
  - **lastpos (n) = {i}**

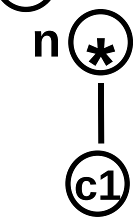
- If n is an or-node (|) with children c1 and c2
  - **lastpos (n) = lastpos(c1) $\cup$ lastpos (c2)**



- If n is an cat-node ($\bullet$) with children c1 and c2
  - **lastpos(n) = If nullable (c2) then lastpos(c1) $\cup$ lastpos (c2)**
  - **else lastpos(c2)**



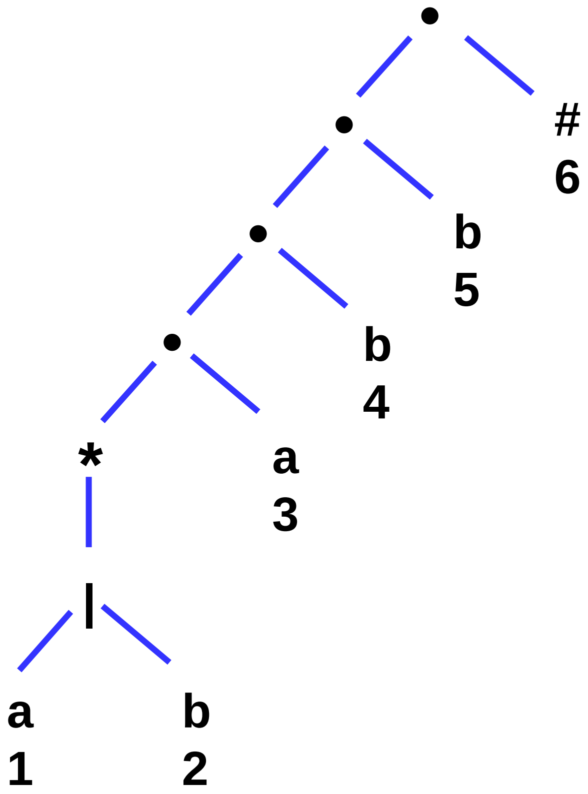
- If n is an star-node (*) with children c1
  - **lastpos (n) = lastpos(c1)**

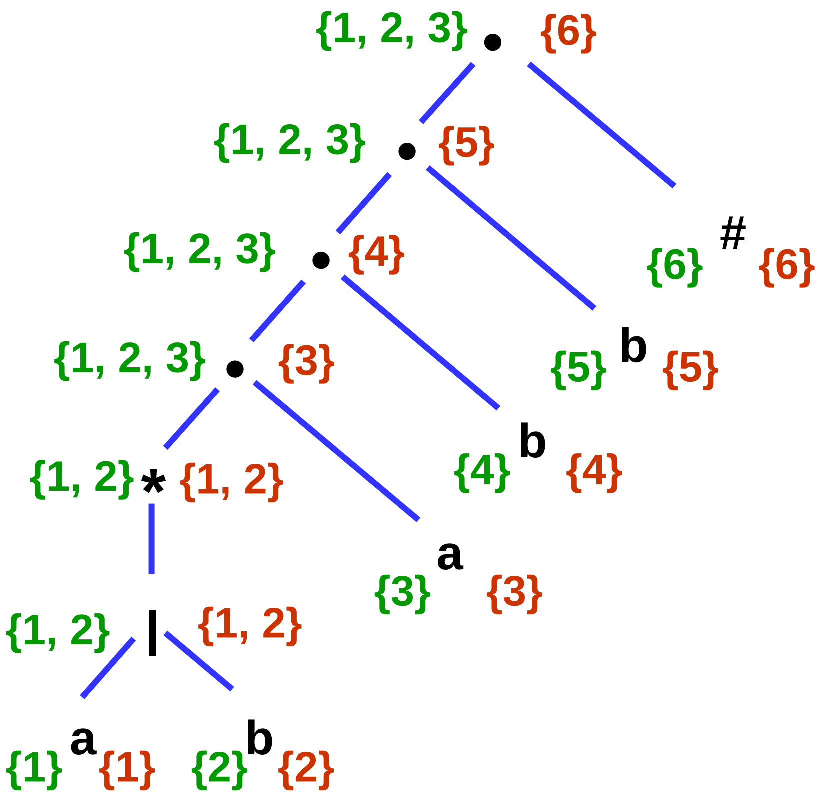


How to evaluate firstpos, lastpos, nullable

<u>n</u>	<u>nullable(n)</u>	<u>firstpos(n)</u>	<u>lastpos(n)</u>
leaf labeled ε	true	Φ	Φ
leaf labeled with position i	false	{i}	{i}
$\mid$ 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)
$\bullet$ c_1 c_2	nullable(c_1) and nullable(c_2)	if (nullable(c_1)) firstpos(c_1) $\cup$ firstpos(c_2) else firstpos(c_1)	if (nullable(c_2)) lastpos(c_1) $\cup$ lastpos(c_2) else lastpos(c_2)
$*$ c_1	true	firstpos(c_1)	lastpos(c_1)

firstpos and *lastpos* example





followpos

Then we define the function followpos for the positions (positions assigned to leaves).

followpos(i) : is the set of positions which can follow the position i in the strings generated by the augmented regular expression.

For example, $(a \mid b)^{*} a \;\#$

1 2 3 4

followpos(1) = {1,2,3}
followpos(2) = {1,2,3}
followpos(3) = {4}
followpos(4) = {}

followpos is just defined for leaves, it is not defined for inner nodes.

Terminology

- Followpos(*i*):
  - Tells what positions can follow position i in the syntax tree

- **Rule 1:**

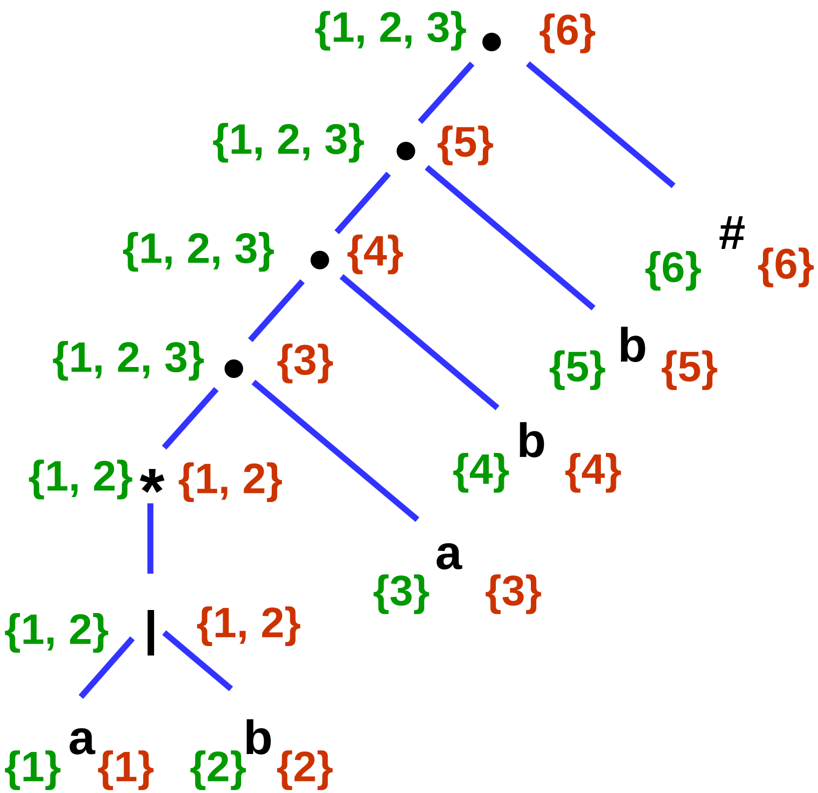
  If ***n*** is a cat-node with left child ***c1*** and right child ***c2*** and ***i*** is a position in lastpos (***c1***), then all positions in firstpos(***c2***) are in followpos(*i*)

- **Rule 2:**

  If ***n*** is a star node, and ***i*** is a position in lastpos(***n***), then all positions in firstpos(***n***) are in followpos(***i***)

- After computing firstpos and lastpos for each node follow pos of each position can be computed by making depth-first traversal of the syntax tree

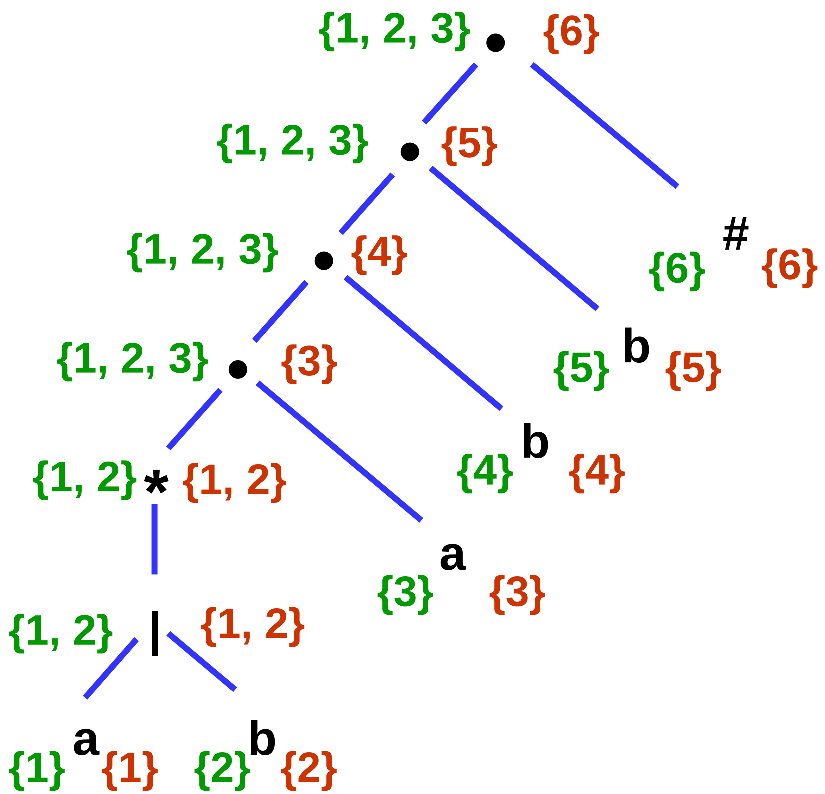
followpos example



Node	followpos
1	{1,2,
2	{1,2,
3	{
4	{
5	{
6	{

- At star-node:
  - *lastpos*(*) = {1,2} and *firstpos*(*)={1,2}
  - According to Rule 2:
    - » followpos{1} = {1,2}
    - » followpos{2} = {1,2}

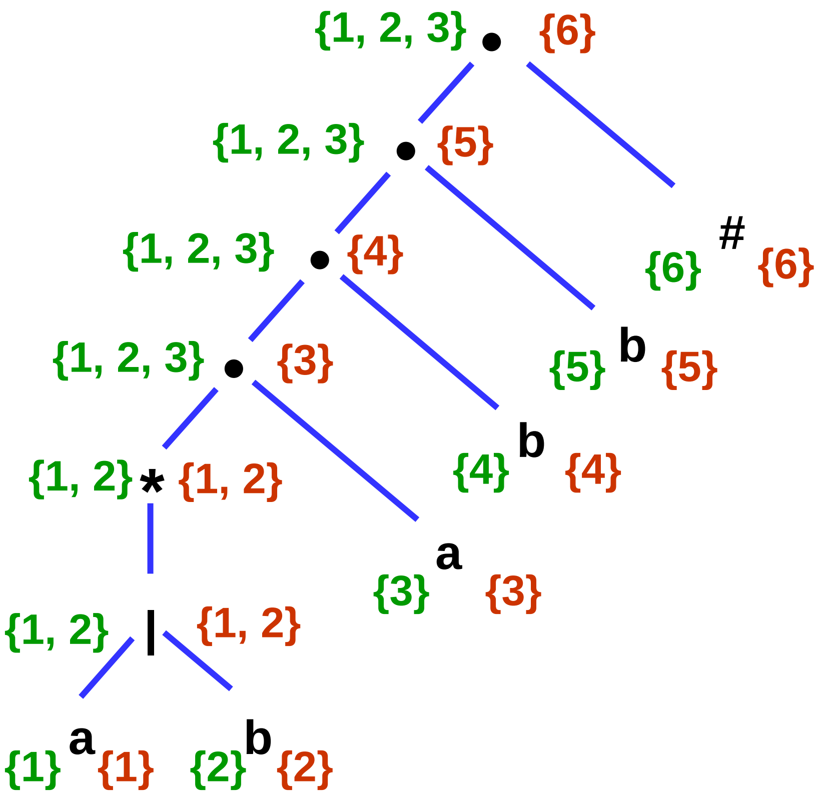
followpos example



Node	Followpos
1	{1,2,3
2	{1,2,3
3	{
4	{
5	{
6	{

- At cat-node above the star-node, ‘*’ is left child and ‘a’ is right child
  - *lastpos*(*) = {1,2} and *firstpos(a)*={3}
  - According to Rule 1:
    - » followpos{1} = {3}
    - » followpos{2} = {3}

followpos example



Node	Followpos
1	{1,2,3
2	{1,2,3
3	{4
4	{5
5	{6
6	{

- At next cat-node '•' is left child and 'b' is right child
  - *lastpos*(•) = {3} and *firstpos(b)* = {4}
  - According to Rule 1:
    - followpos{3} = {4}
- Similarly, followpos{4}={5} and followpos{5}={6}

Construction of DFA from RE

- Algorithm

Initially, the only unmarked state in **Dstates** is ***firstpos(*root*)***

while there is an unmarked state **T** in **Dstates** do begin

    Mark T;

    For each input symbol **a** do begin

        Let **U** be the set of positions that are in followpos(**p**) for some position **p** in **T** such that the symbol at position **p** is **a**

        If **U** is not empty and is not in **Dstates** then

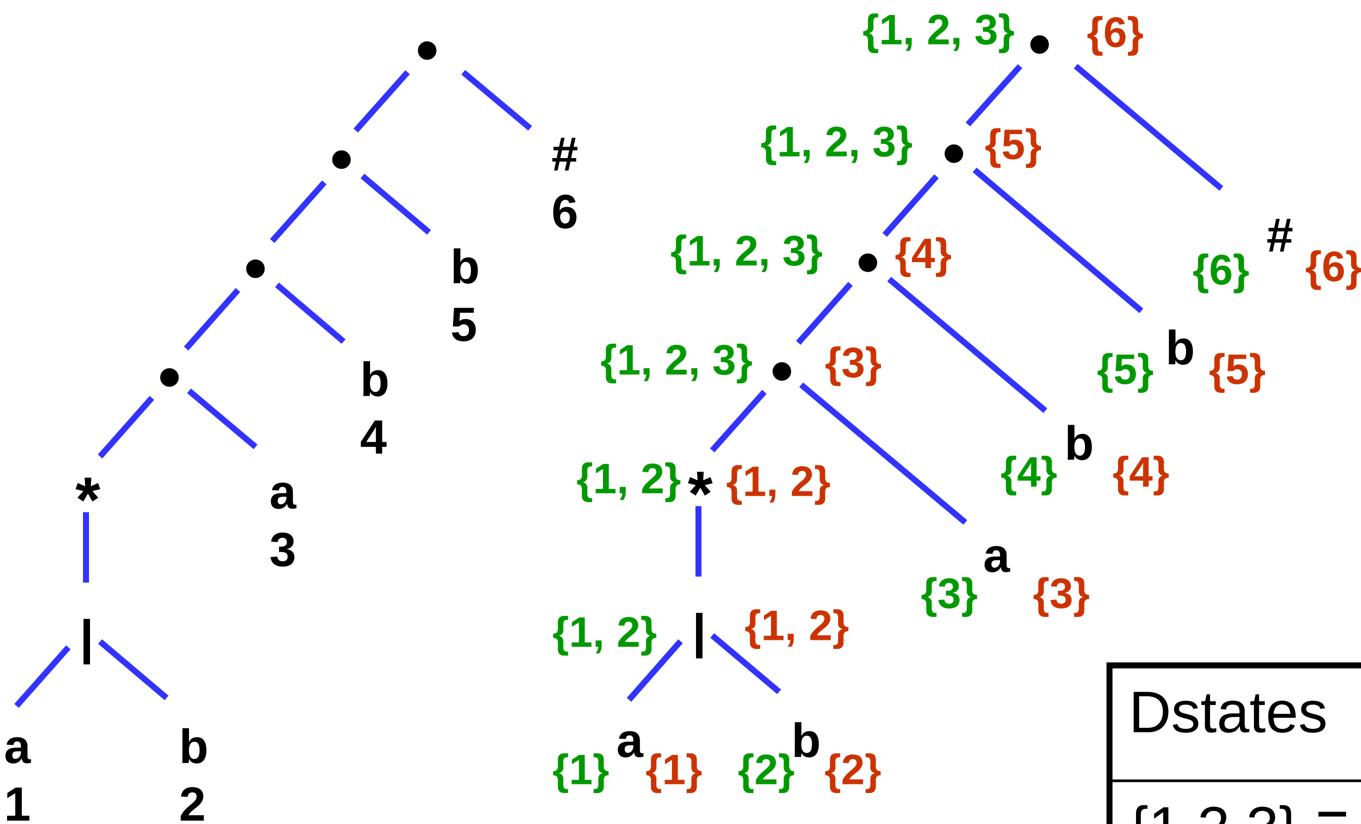
            Add **U** as an unmarked states to **Dstates**

        Dtrans[T,a]=U

    End

end

DFA for (a|b)*abb#



Node	followpos
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	-

firstpos{root} = {1,2,3} ≡ A (unmarked)

For the input symbol a, positions are 1, 3

∴ followpos(1) ∪ followpos{3}

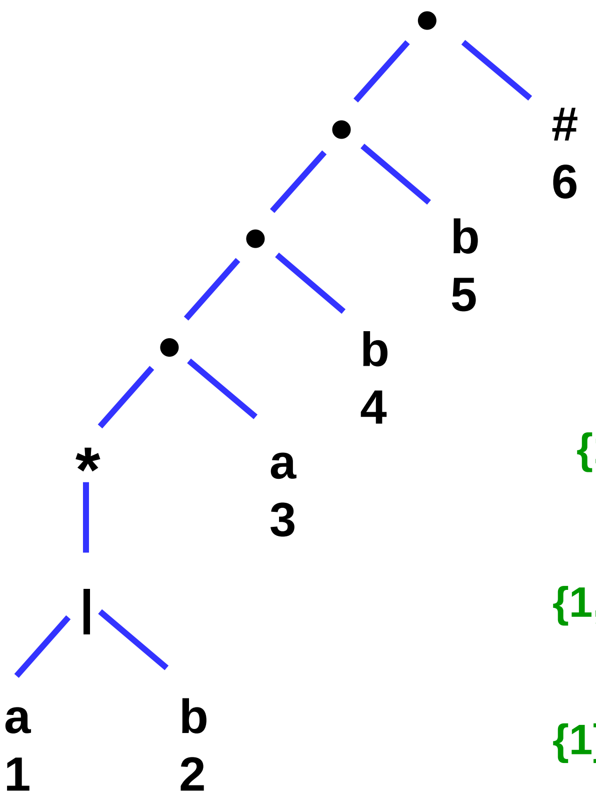
={1,2,3,4} ≡ B

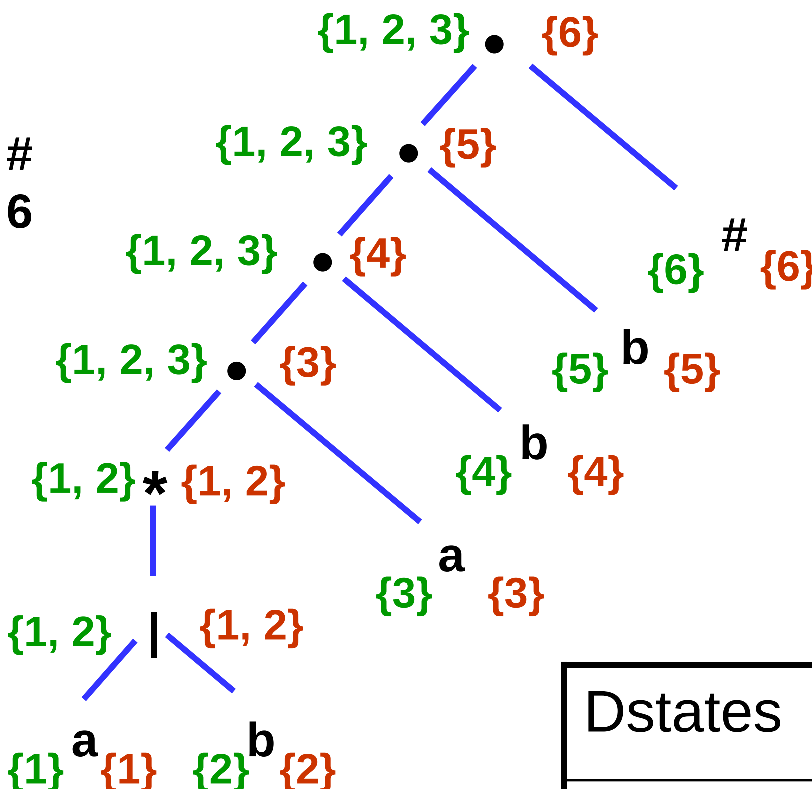
For the input symbol b, positions are 2

∴ followpos(2)= {1,2,3,} ≡ A

Dstates	a	b
{1,2,3} ≡ A	B	A
{1,2,3,4} ≡ B		

DFA for (a|b)*abb#





Node	followpos
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	-

{1,2,3,4} ≡ B (unmarked)

For the input symbol a, positions are 1, 3

∴ followpos(1) ∪ followpos{3}

={1,2,3,4} ≡ B

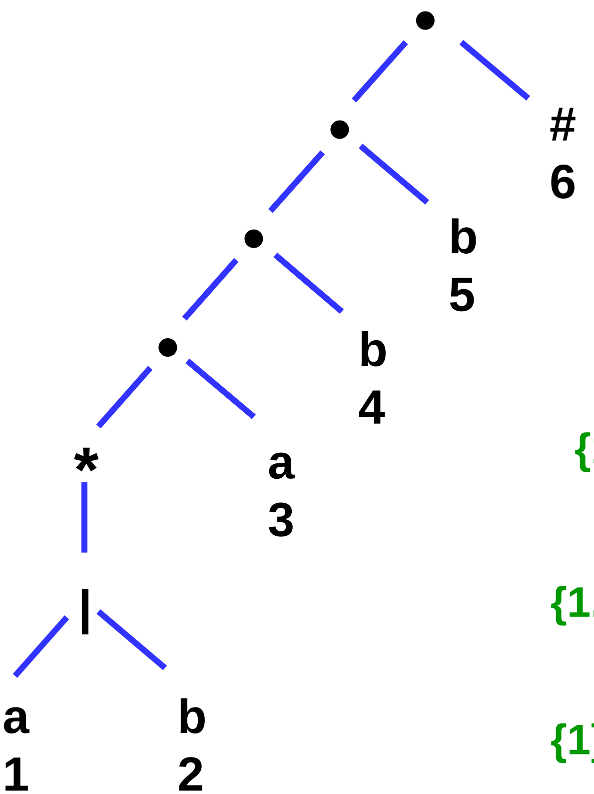
For the input symbol b, positions are 2, 4

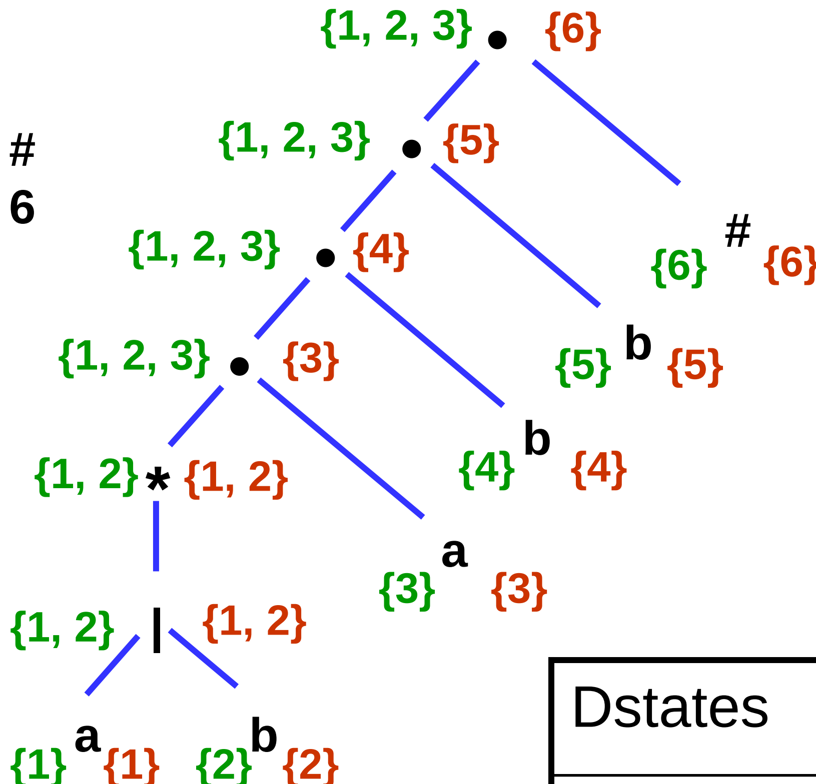
∴ followpos(2) ∪ followpos{4}

= {1,2,3,5} ≡ C

Dstates	a	b
{1,2,3} ≡ A	B	A
{1,2,3,4} ≡ B	B	C
{1,2,3,5} ≡ C		

DFA for (a|b)*abb#





Node	followpos
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	-

{1,2,3,5} ≡ C (unmarked)

For the input symbol a, positions are 1, 3

∴ followpos(1) ∪ followpos{3}

={1,2,3,4} ≡ B

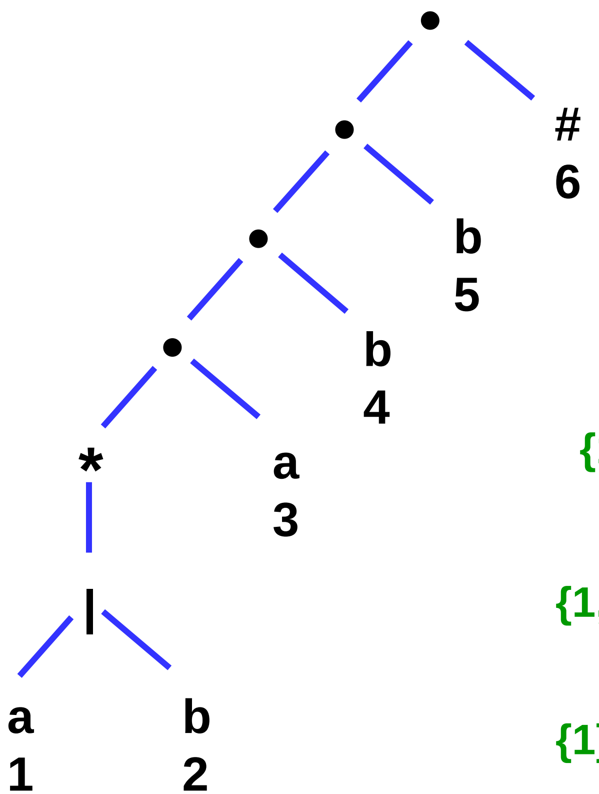
For the input symbol b, positions are 2, 5

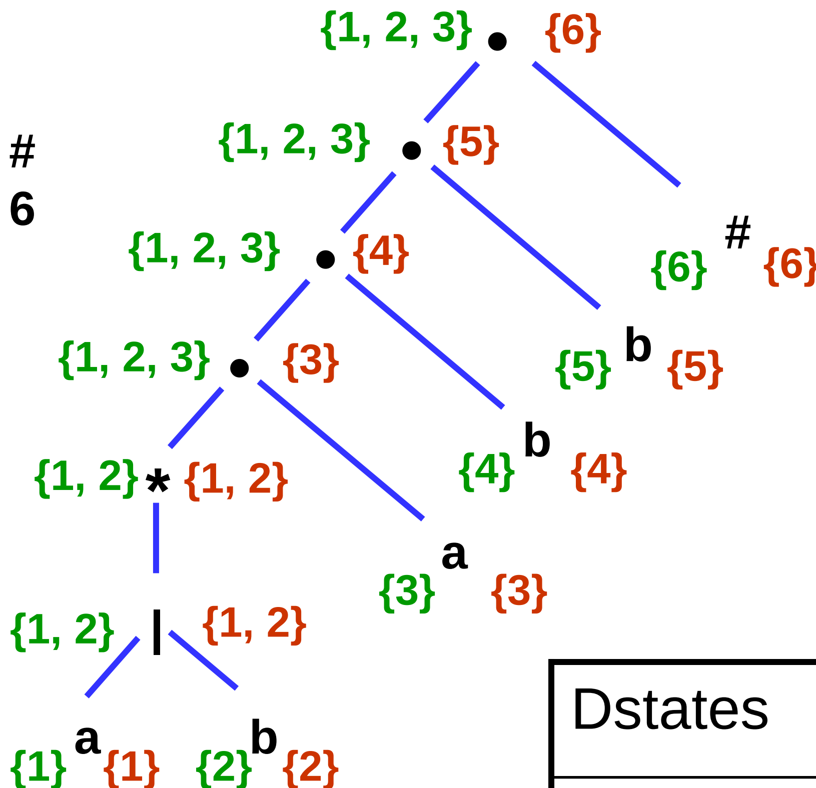
∴ followpos(2) ∪ followpos{5}

= {1,2,3,6} ≡ D

Dstates	a	b
{1,2,3} ≡ A	B	A
{1,2,3,4} ≡ B	B	C
{1,2,3,5} ≡ C	B	D
{1,2,3,6} ≡ D		

DFA for (a|b)*abb#





Node	followpos
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	-

{1,2,3,6} ≡ D (unmarked)

For the input symbol a, positions are 1, 3

∴ followpos(1) ∪ followpos{3}

={1,2,3,4} ≡ B

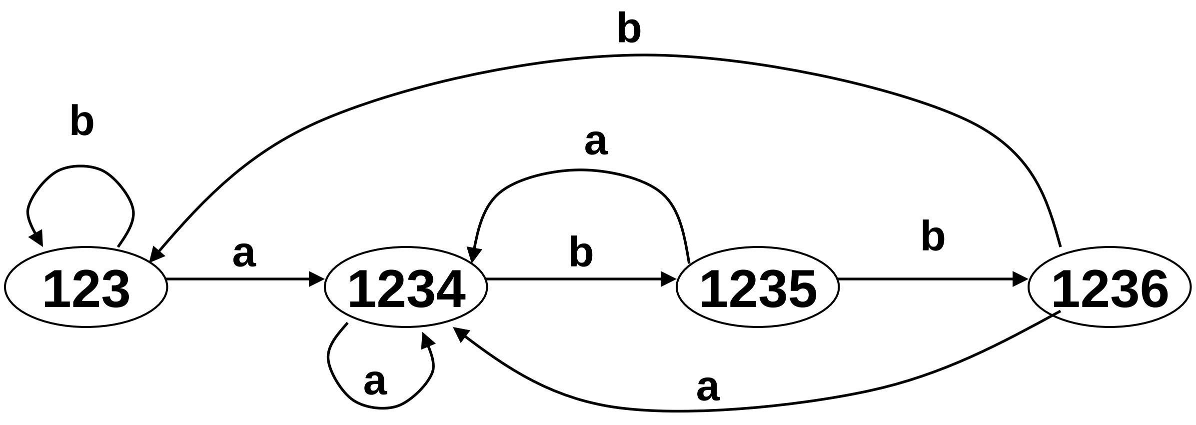
For the input symbol b, positions are 2

∴ followpos(2)

= {1,2,3} ≡ A

Dstates	a	b
{1,2,3} ≡ A	B	A
{1,2,3,4} ≡ B	B	C
{1,2,3,5} ≡ C	B	D
{1,2,3,6} ≡ D	B	A

DFA for (a|b)*abb#



Simulation of a FA

```
s ← s0
c ← nextchar;
while c ≠ eof do
   s ← move(s,c);
   c ← nextchar;
end;
if s is in F then return "yes"
             else return "no"
```

DFA simulation

```
S ← ∈-closure({s0})
c ← nextchar;
while c ≠ eof do
   S ← ∈-closure(move(S,c));
   c ← nextchar;
end;
if S∩F≠∅ then return "yes"
             else return "no"
```

NFA simulation

End of slide