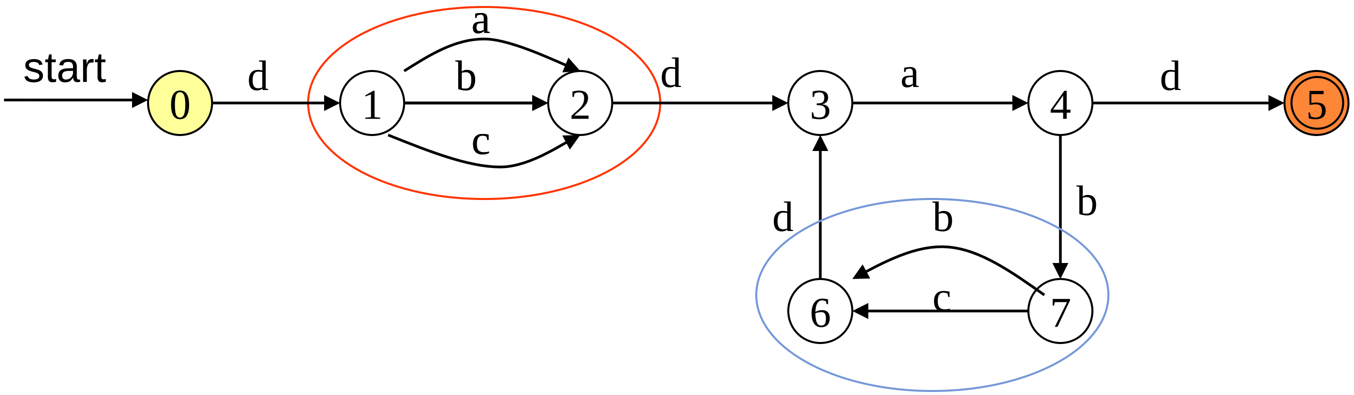
CSE 4102
Lexical Analysis

Lecture 03

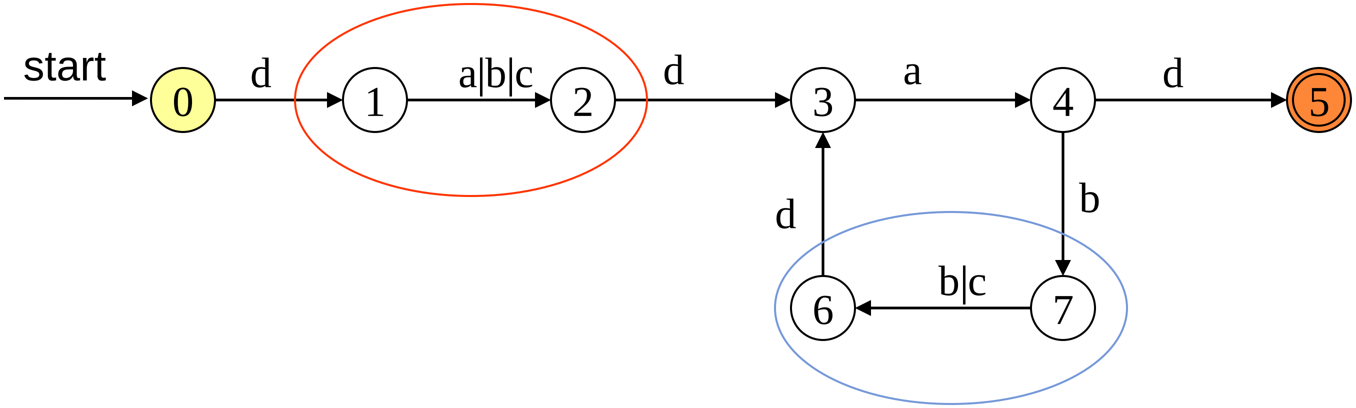
Converting DFAs to REs

1. Combine serial links by concatenation
2. Combine parallel links by alternation
3. Remove self-loops by Kleene closure
4. Select a node (other than initial or final) for removal. Replace it with a set of equivalent links whose path expressions correspond to the in and out links
5. Repeat steps 1-4 until the graph consists of a single link between the entry and exit nodes.

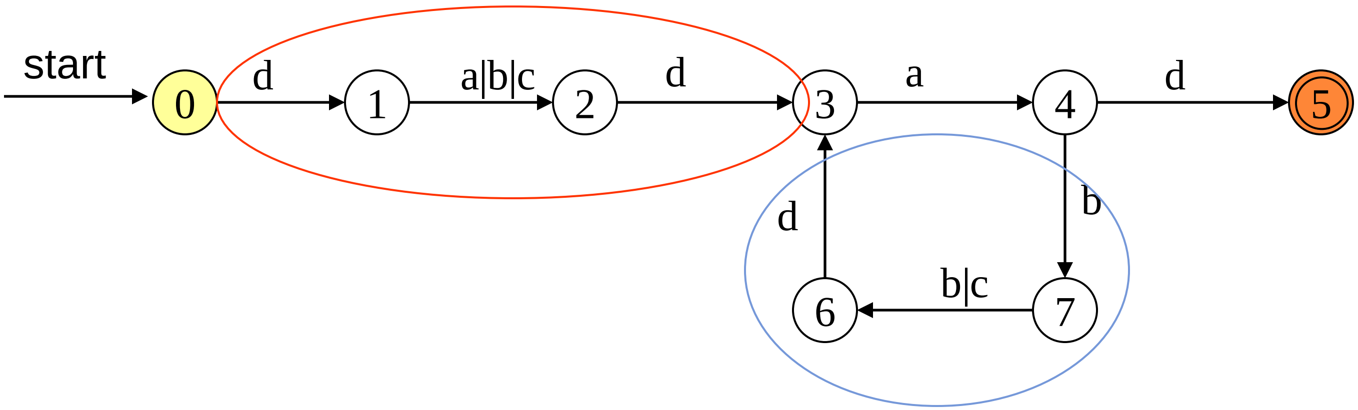
EXAMPLE



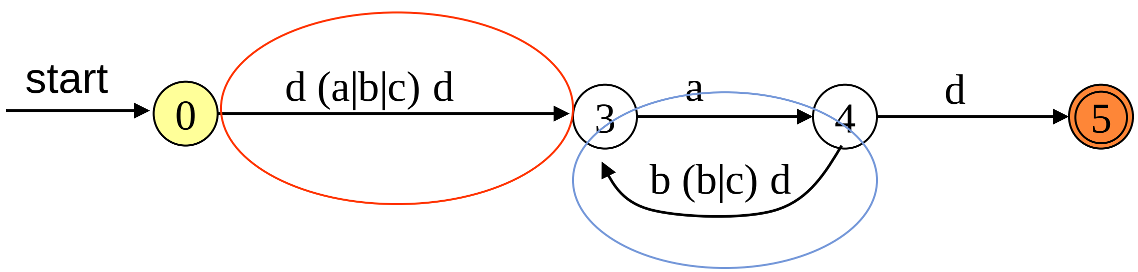
parallel edges become alternation



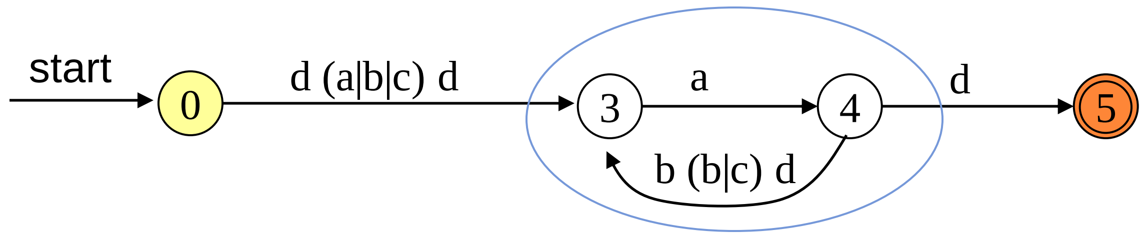
EXAMPLE



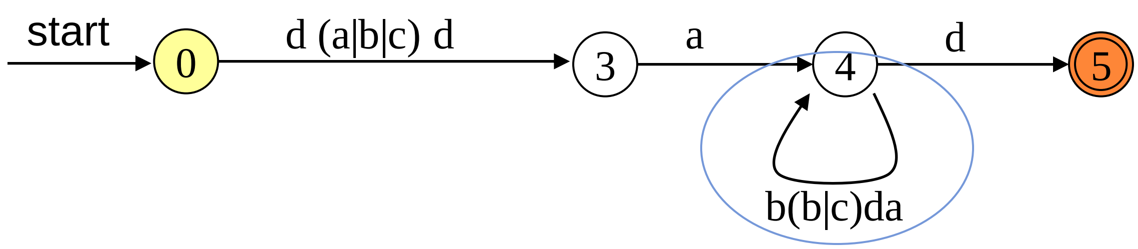
serial edges become concatenation



Example



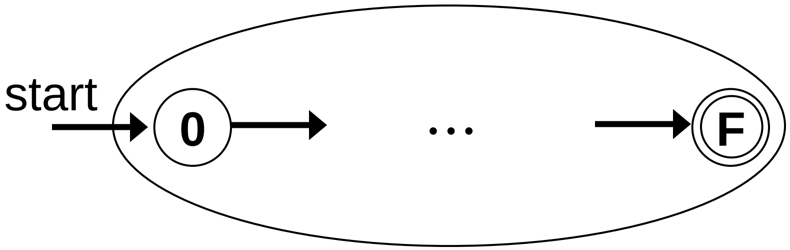
Find paths that can be "shortened"



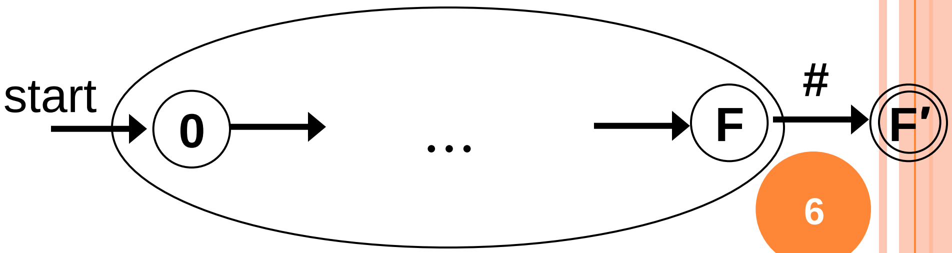
REGULAR EXPRESSION TO DFA

- Important States of NFA
  - If it has a non-ε out-transition
  - *move(s,a)* is non-empty if s is important
  - Accepting states are not important states
    - Adding a unique marker # after the RE r (i.e. r#) we can make the accepting states important
    - Now a state with a transition on # will be accepting state

r

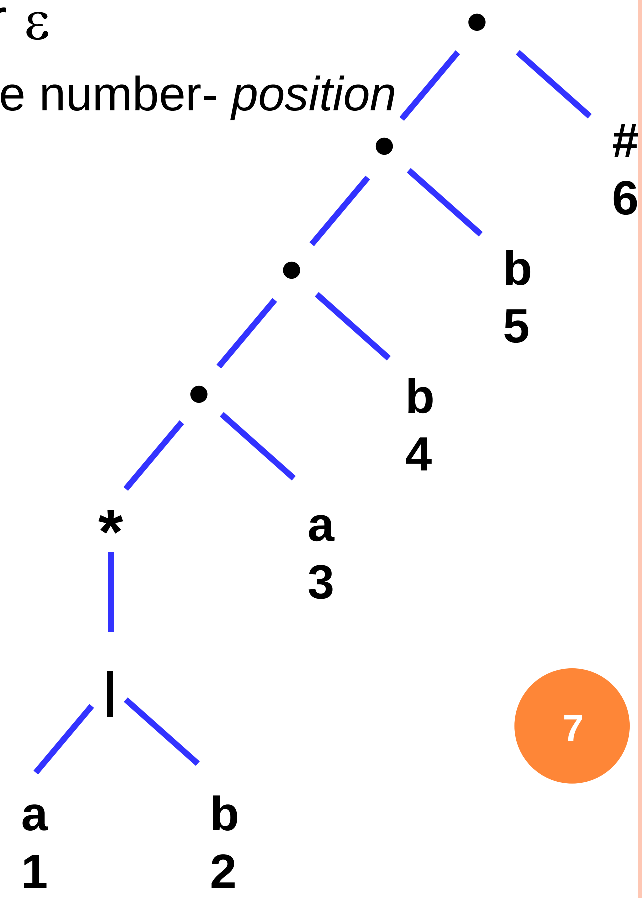


r#

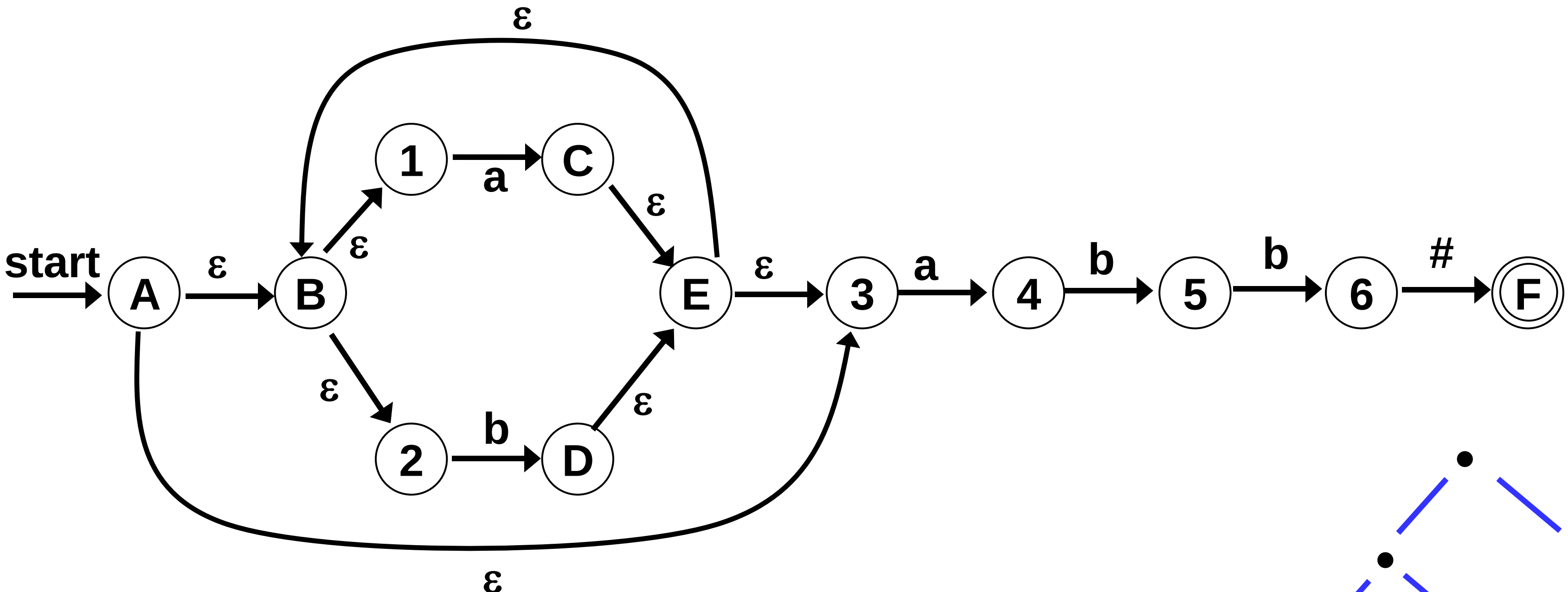


Syntax Tree

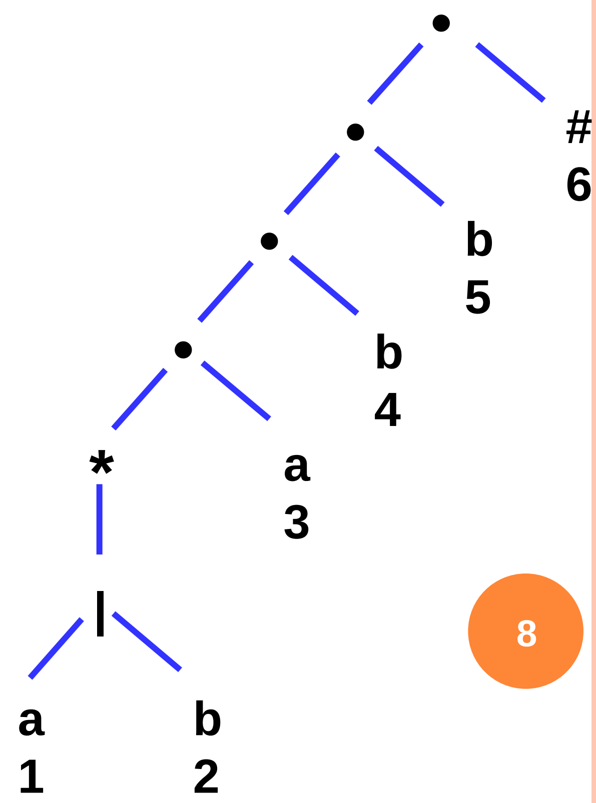
- Augmented RE (r#) can be represented by a syntax tree
  - Leaves contain: Alphabet symbols or ε
    - Each non-ε leaf is associated with a unique number- *position* of the leaf and *position* of the symbol
  - Internal nodes contain: Operators
    - *cat-node*, *or-node* or *star-node*
- Syntax tree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



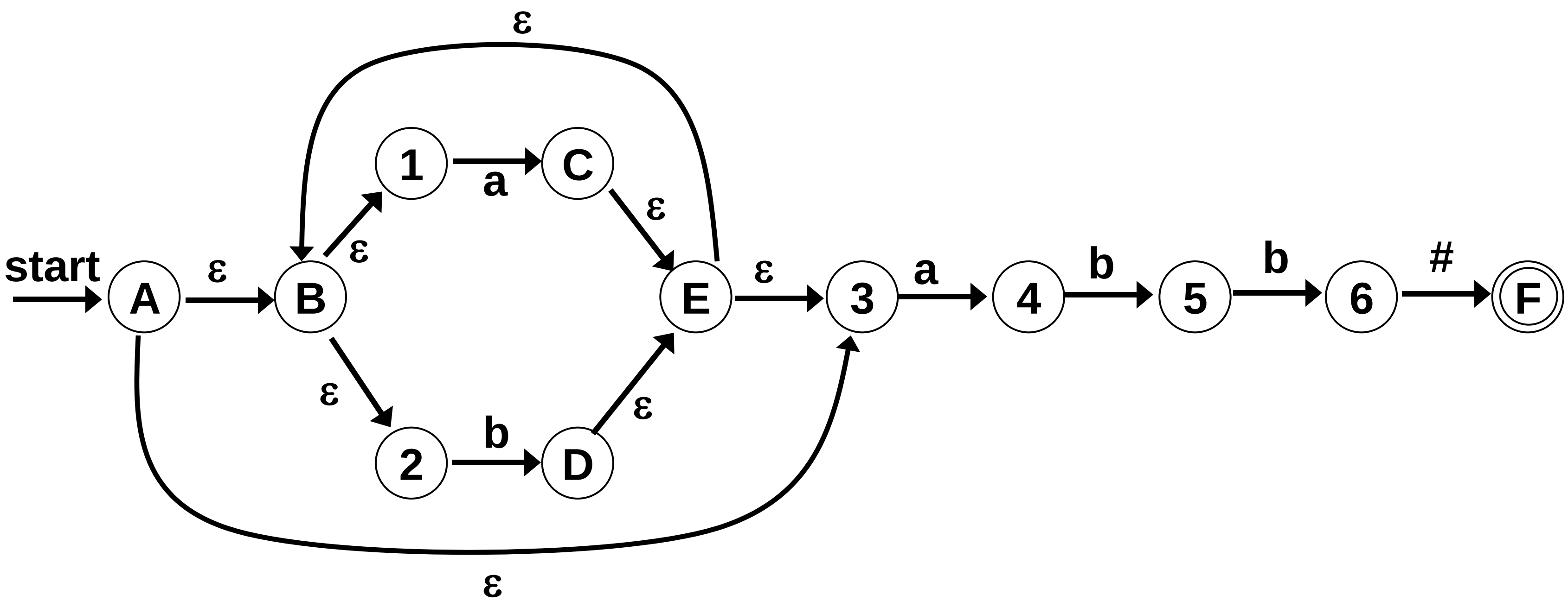
NFA FOR (A|B)*ABB#

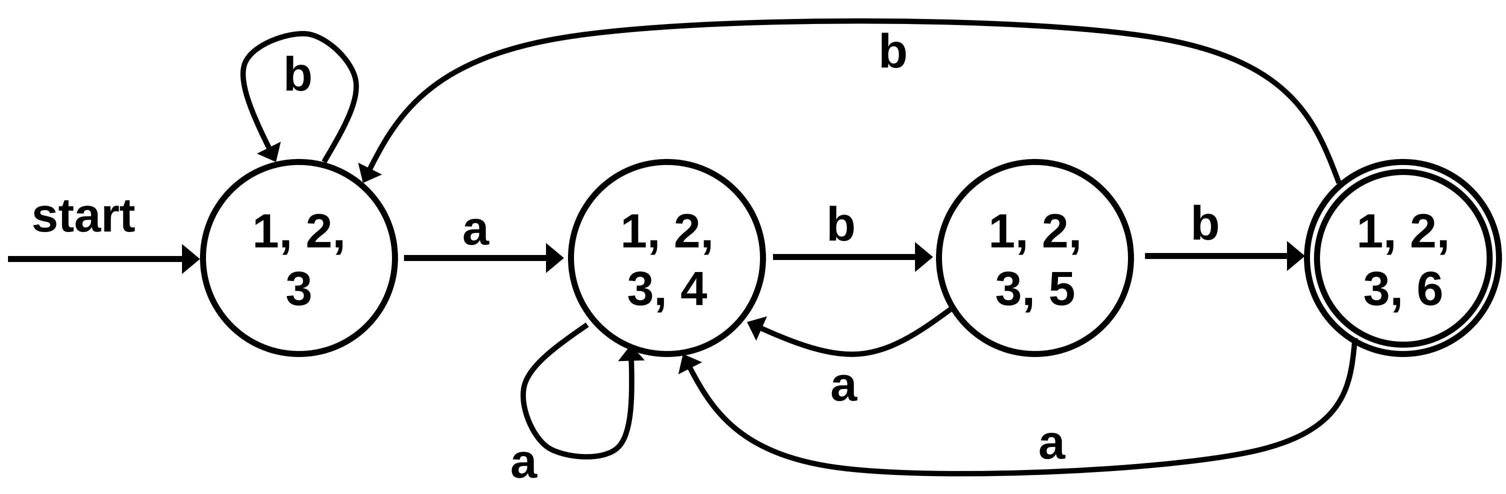


- Lettered states are non-important states
- Number states are important states
  - Numbers correspond to the number in syntax tree



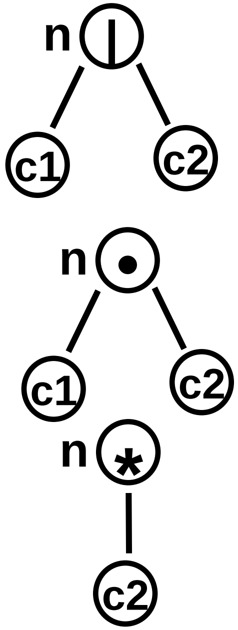
DFA for (a|b)*abb#





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

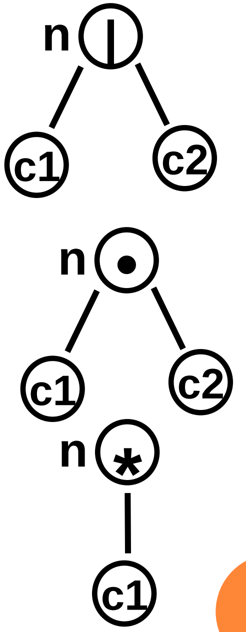
- If n is a leaf labeled by ε then
  - **firstpos (n) = ∅**

- 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) ∪ firstpos (c2)**

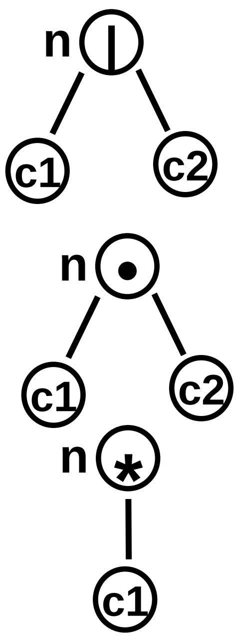
- If n is a cat-node (•) with children c1 and c2
  - **firstpos(n) = If nullable (c1) then firstpos(c1) ∪ firstpos (c2)**
    else firstpos(c1)

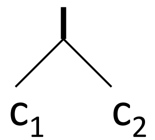
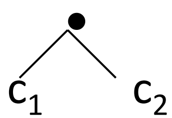
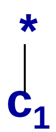
- If n is an star-node (*) with children c1
  - **firstpos (n) = firstpos(c1)**



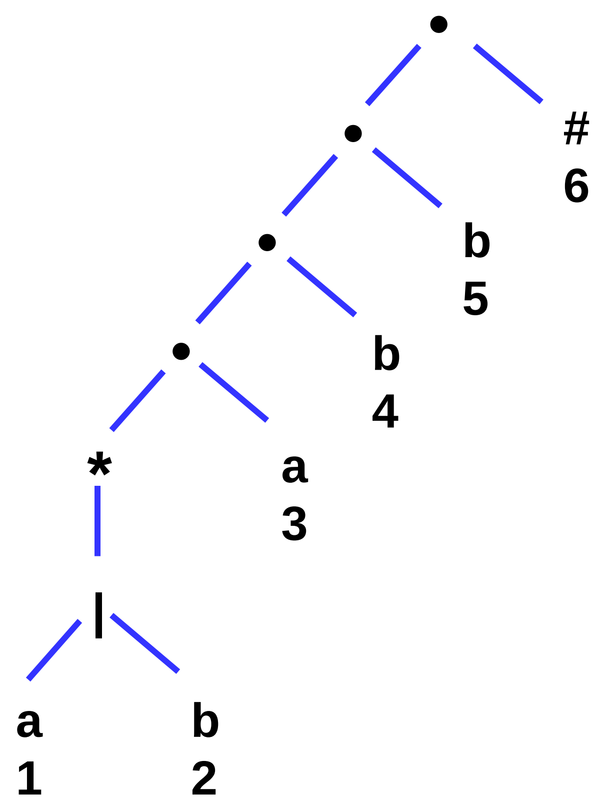
TERMINOLOGY

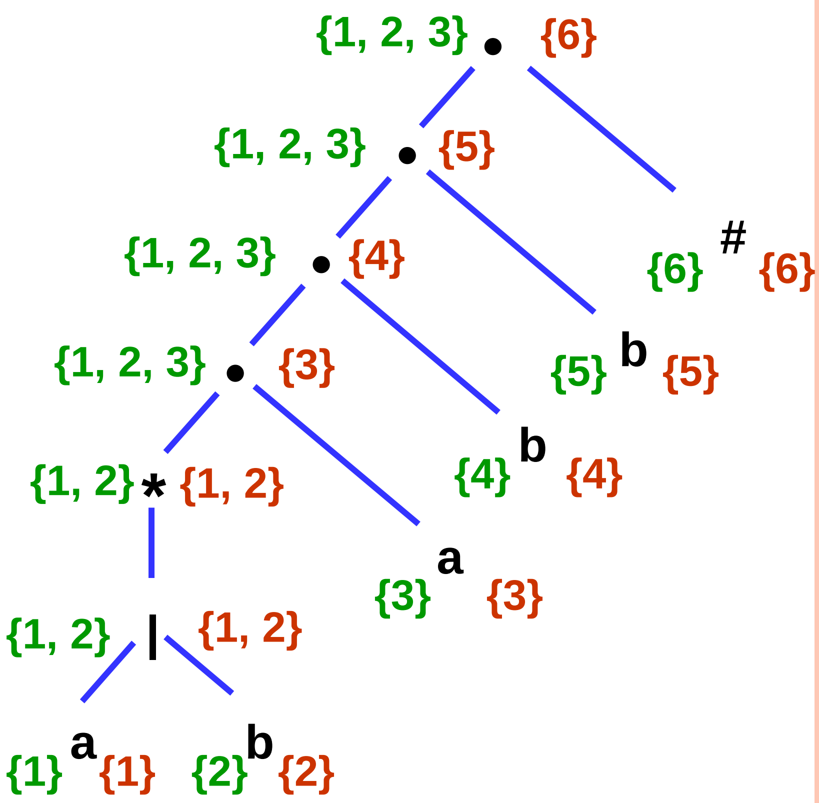
- Lastpos(n):
  - Set of positions that can match the last symbol of a string generated by the sub-expression rooted at n
- If n is a leaf labeled by ε then
  - **lastpos (n) = ∅**
- If n is a leaf labeled with position *i*
  - **lastpos (n) = {i}**
- If n is an or-node (|) with children c1 and c2
  - **lastpos (n) = lastpos(c1) ∪ lastpos (c2)**
- If n is an cat-node (•) with children c1 and c2
  - **lastpos(n) = If nullable (c2) then lastpos(c1) ∪ lastpos (c2)**
  - **else lastpos(c2)**
- If n is an star-node (*) with children c1
  - **lastpos (n) = lastpos(c1)**



n	nullable(n)	firstpos(n)	lastpos(n)
leaf labeled ε	true	Φ	Φ
leaf labeled with position i	false	{i}	{i}
	nullable(c_1) or nullable(c_2)	firstpos(c_1) $\cup$ firstpos(c_2)	lastpos(c_1) $\cup$ lastpos(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)
	**true**	**firstpos(c_1)**	**lastpos(c_1)**

FIRSTPOS AND *LASTPOS* EXAMPLE





TERMINOLOGY

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

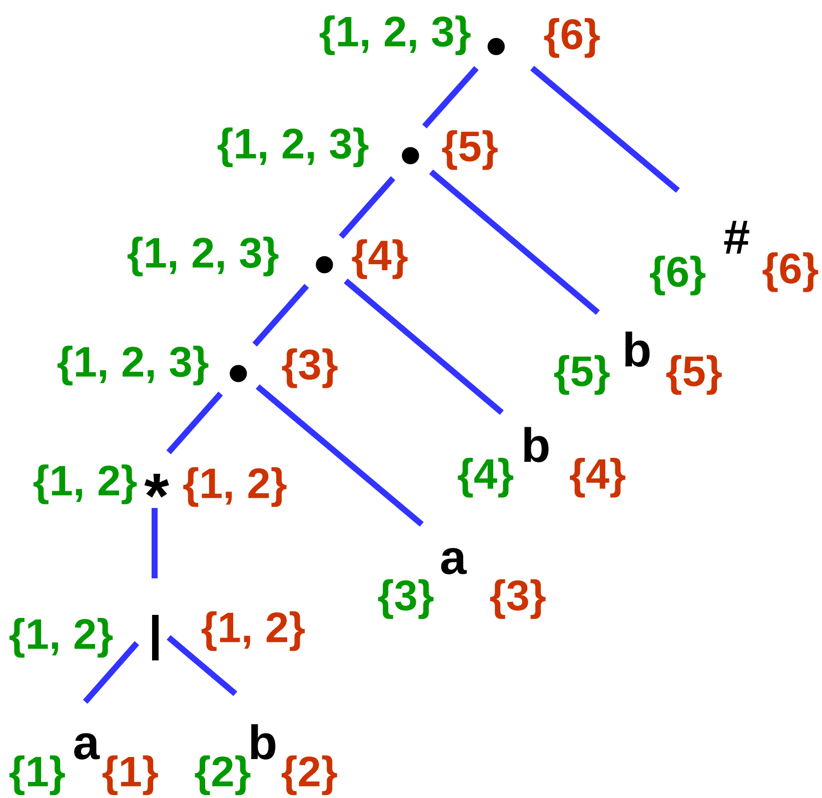
- **Rule 1:**

  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***)

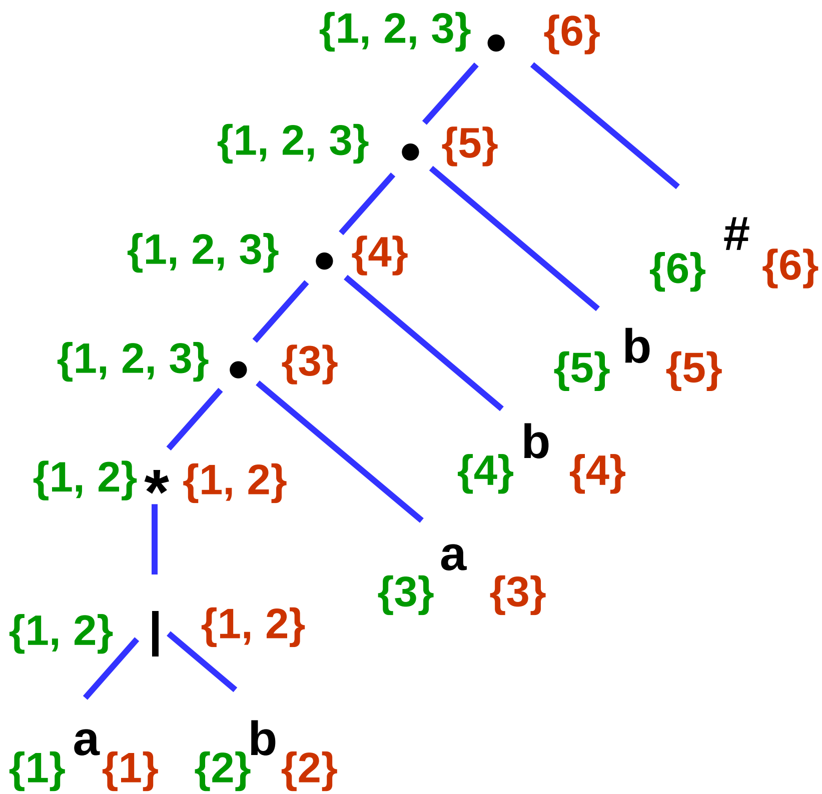
FOLLOWPOS EXAMPLE



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}

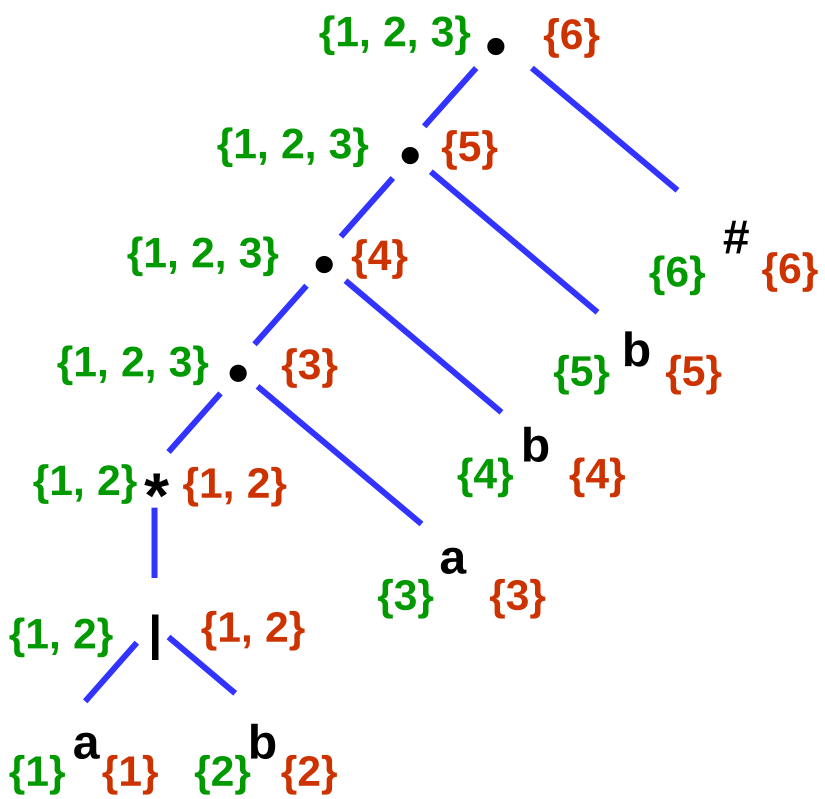
FOLLOWPOS EXAMPLE



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}

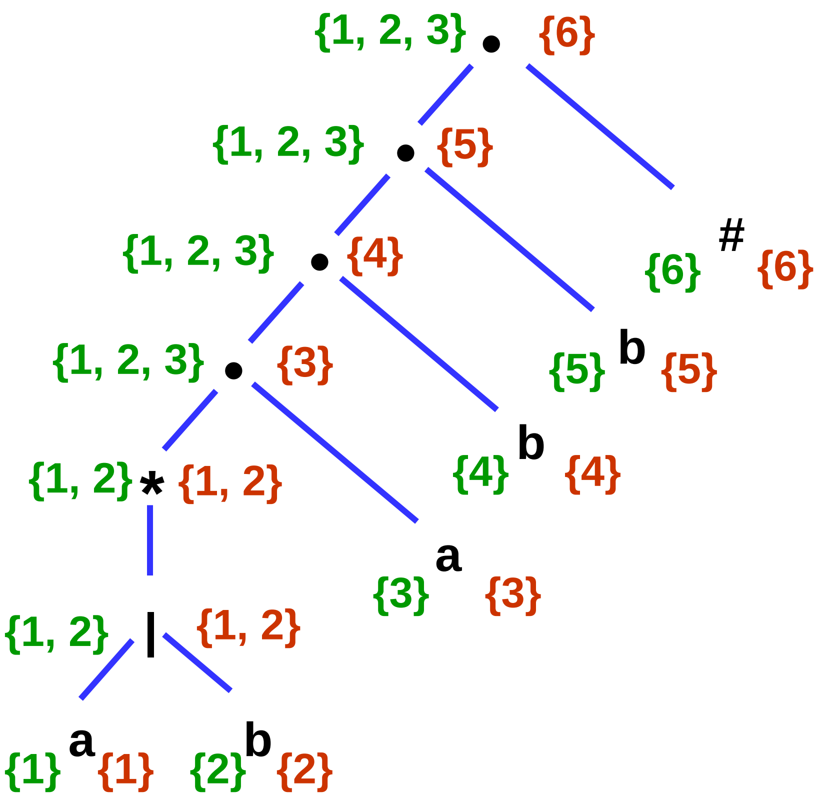
FOLLOWPOS EXAMPLE



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}

FOLLOWPOS EXAMPLE



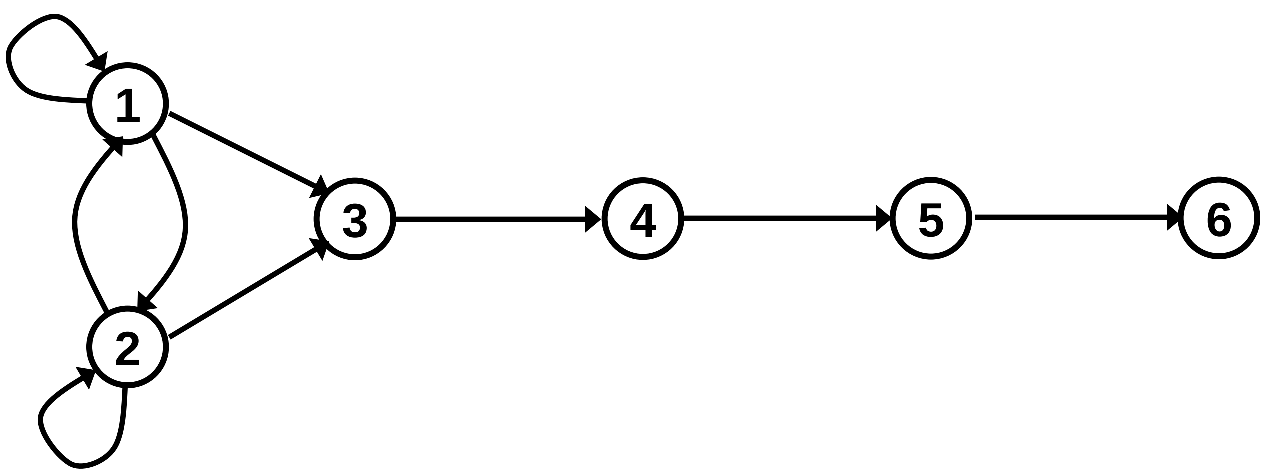
Node	followpos
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	-

FOLLOWPOS GRAPH

- A node for each position
- Edge from node i to node j if j $\in$ followpos{i}

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

- *followpos* graph becomes equivalent NFA without ε-transition if
  - All positions in *firstpos* of root become start state
  - Label edge {i,j} by the symbol at position j
  - Position associated with # only accepting state



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

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

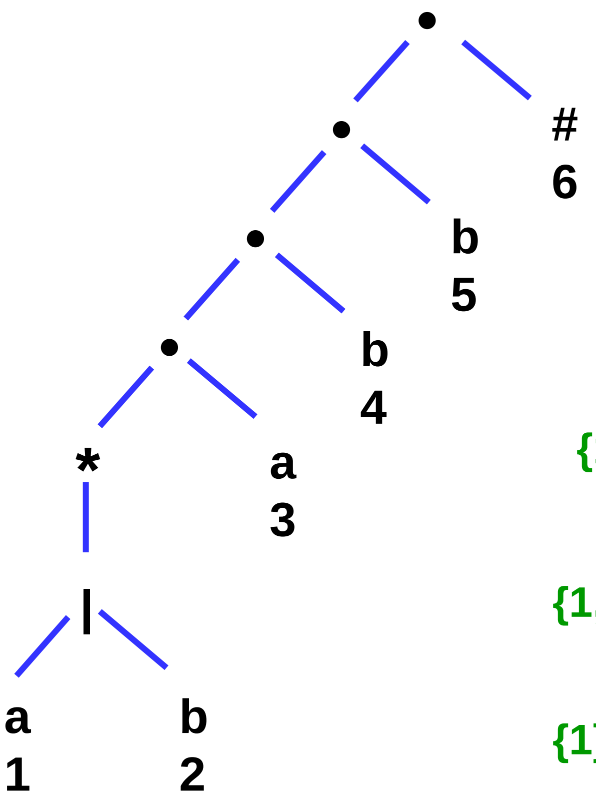
            Add **U** as an unmarked states to **Dstates**

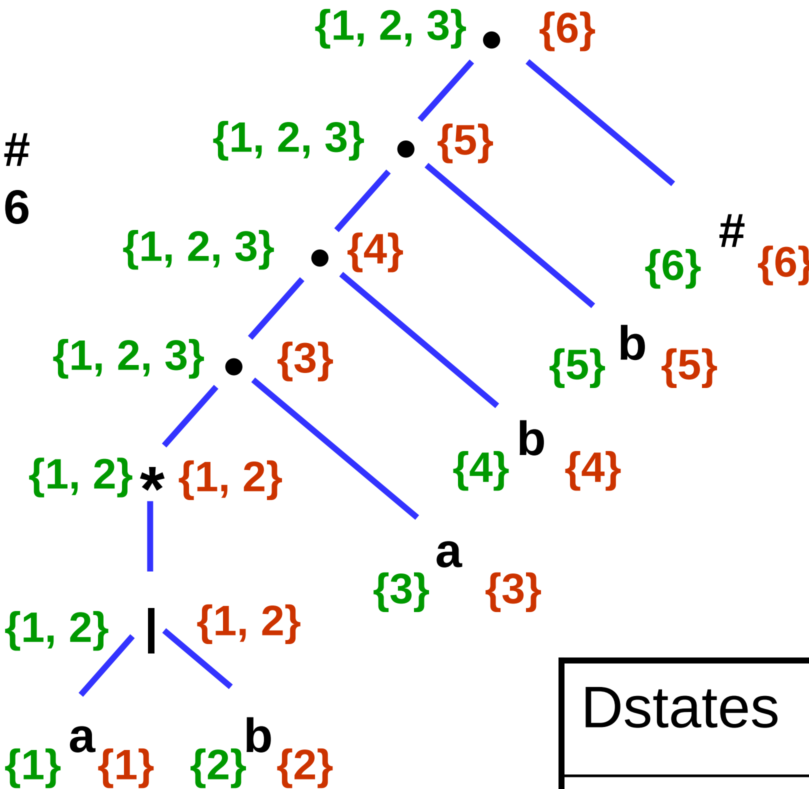
        Dtrans[T,a]=U

    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}

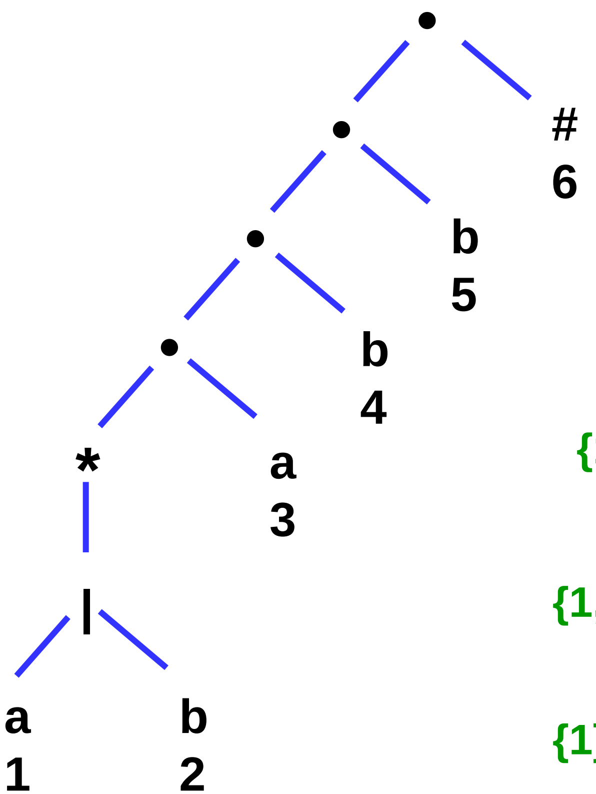
={1,2,3,4} ≡ B

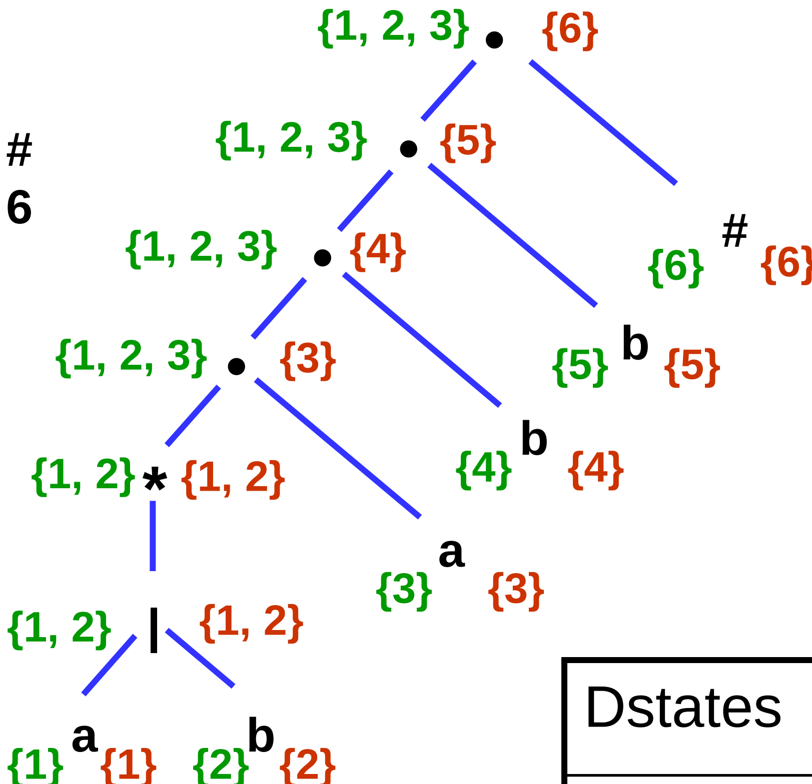
For the input symbol **b,** positions are 2

∴ 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

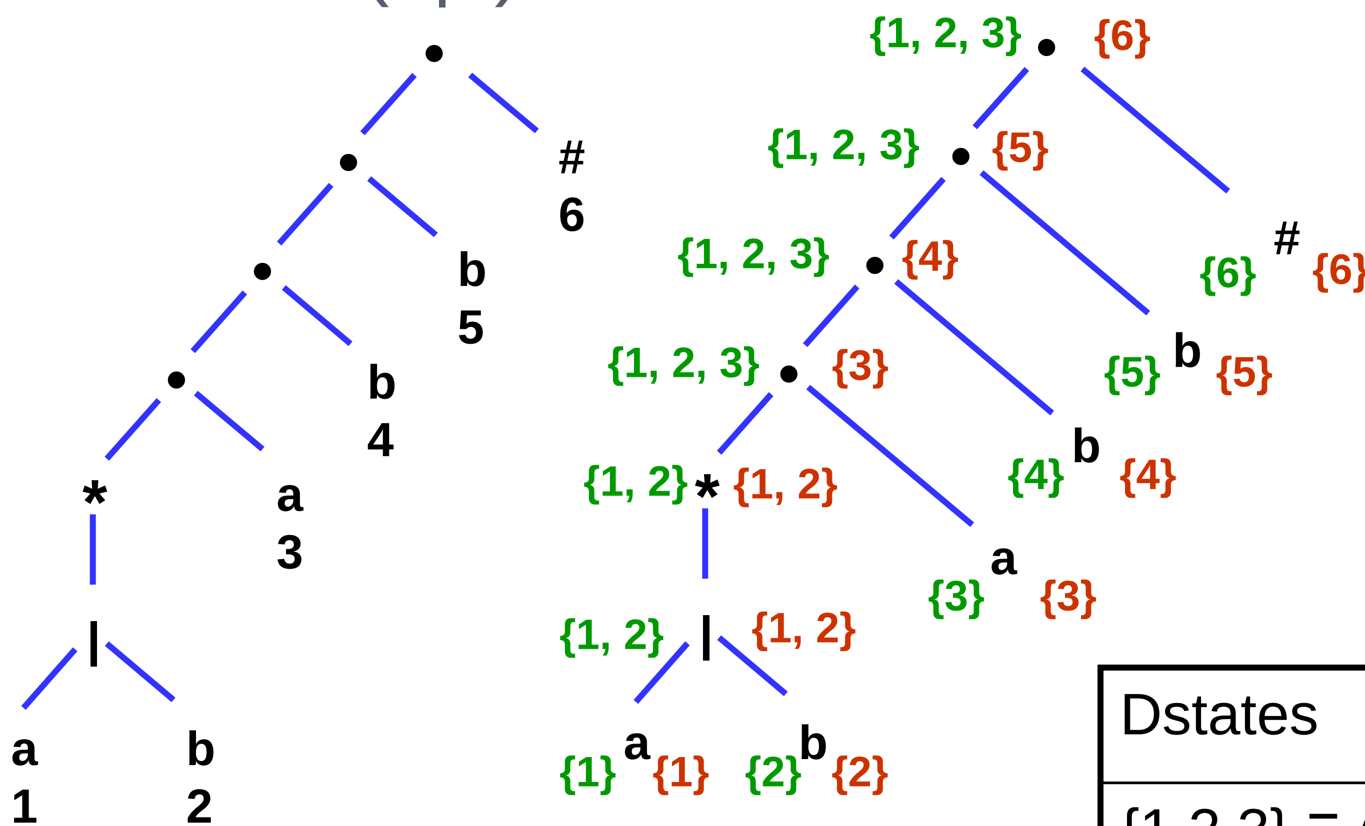
For the input symbol **b,** positions are 2, 4

∴ 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

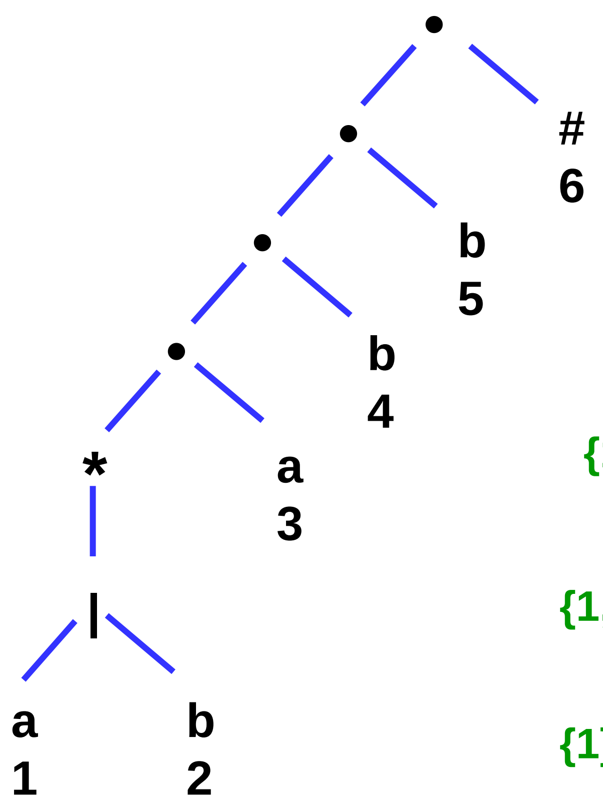
For the input symbol **b,** positions are 2, 5

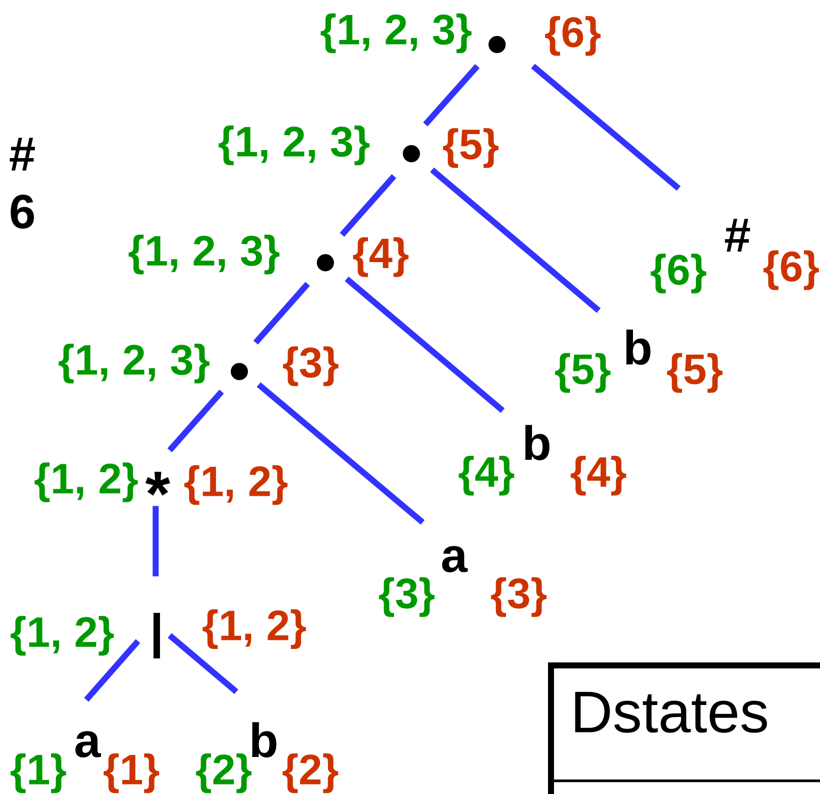
∴ followpos(2) ∪ followpos{5}

= {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

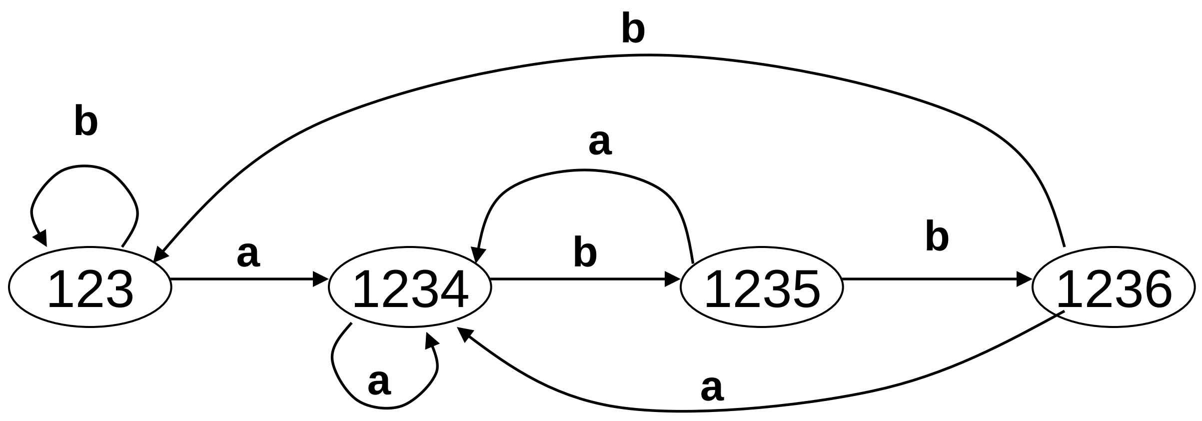
For the input symbol **b,** positions are 2

∴ 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#



Thank You