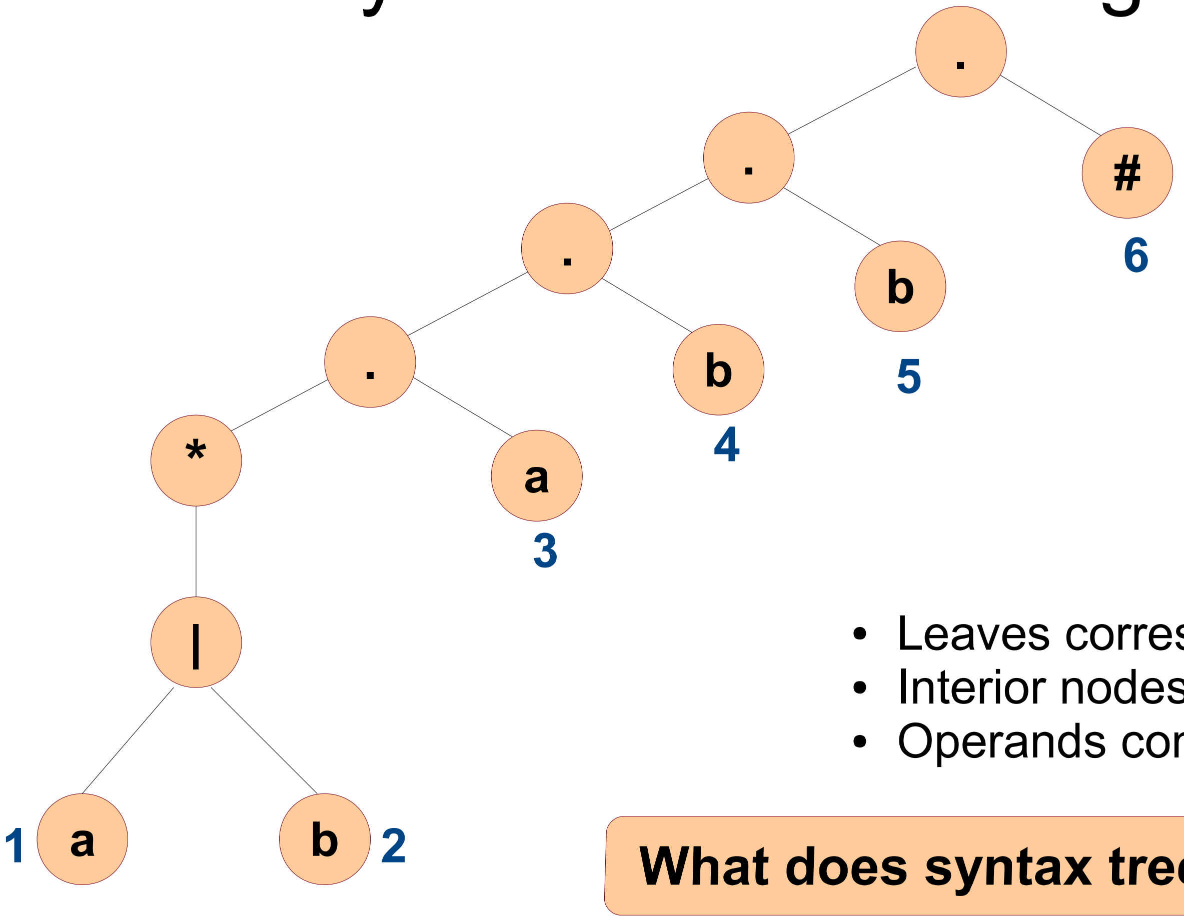
Regex → DFA

- Regex is (a|b)*abb**#**.
- Construct a syntax tree for the regex.



- Leaves correspond to operands.
- Interior nodes correspond to operators.
- Operands constitute strings.

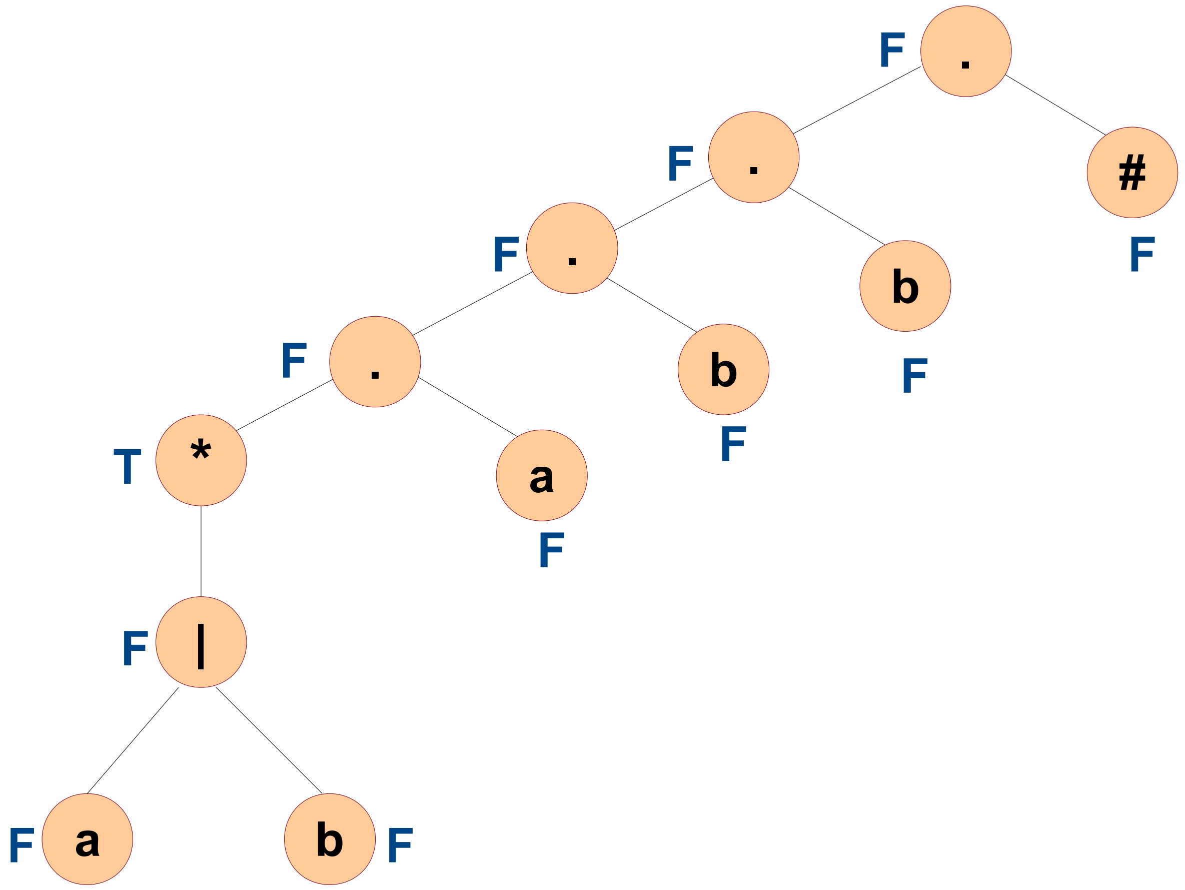
What does syntax tree for regex indicate?

Functions from Syntax Tree

- For a syntax tree node n
  - *nullable*(n): true if n represents ϵ.
  - *firstpos*(n): set of positions that correspond to the first symbol of strings in n's subtree.
  - *lastpos*(n): set of positions that correspond to the last symbol of strings in n's subtree.
  - *followpos*(n): set of next possible positions from n for valid strings.

nullable

- Regex is (a|b)*abb#.



nullable

Node n	**nullable(n)**
leaf labeled ϵ	true
leaf with position i	false
or-node n = c1 | c2	nullable(c1) or nullable(c2)
cat-node n = c1c2	nullable(c1) and nullable(c2)
star-node n = c*	true

Classwork: Write down the rules for firstpos(n).

firstpos

Node n	**firstpos(n)**
leaf labeled ϵ	{ }
leaf with position i	{i}
or-node n = c1 | c2	firstpos(c1) U firstpos(c2)
cat-node n = c1c2	
star-node n = c*	firstpos(c)

firstpos

Node n	**firstpos(n)**
leaf labeled ϵ	{ }
leaf with position i	{i}
or-node n = c1 | c2	firstpos(c1) U firstpos(c2)

cat-node n = c1c2	if (nullable(c1)) firstpos(c1) U firstpos(c2)
	else firstpos(c1)

star-node n = c*	firstpos(c)

Classwork: Write down the rules for lastpos(n).

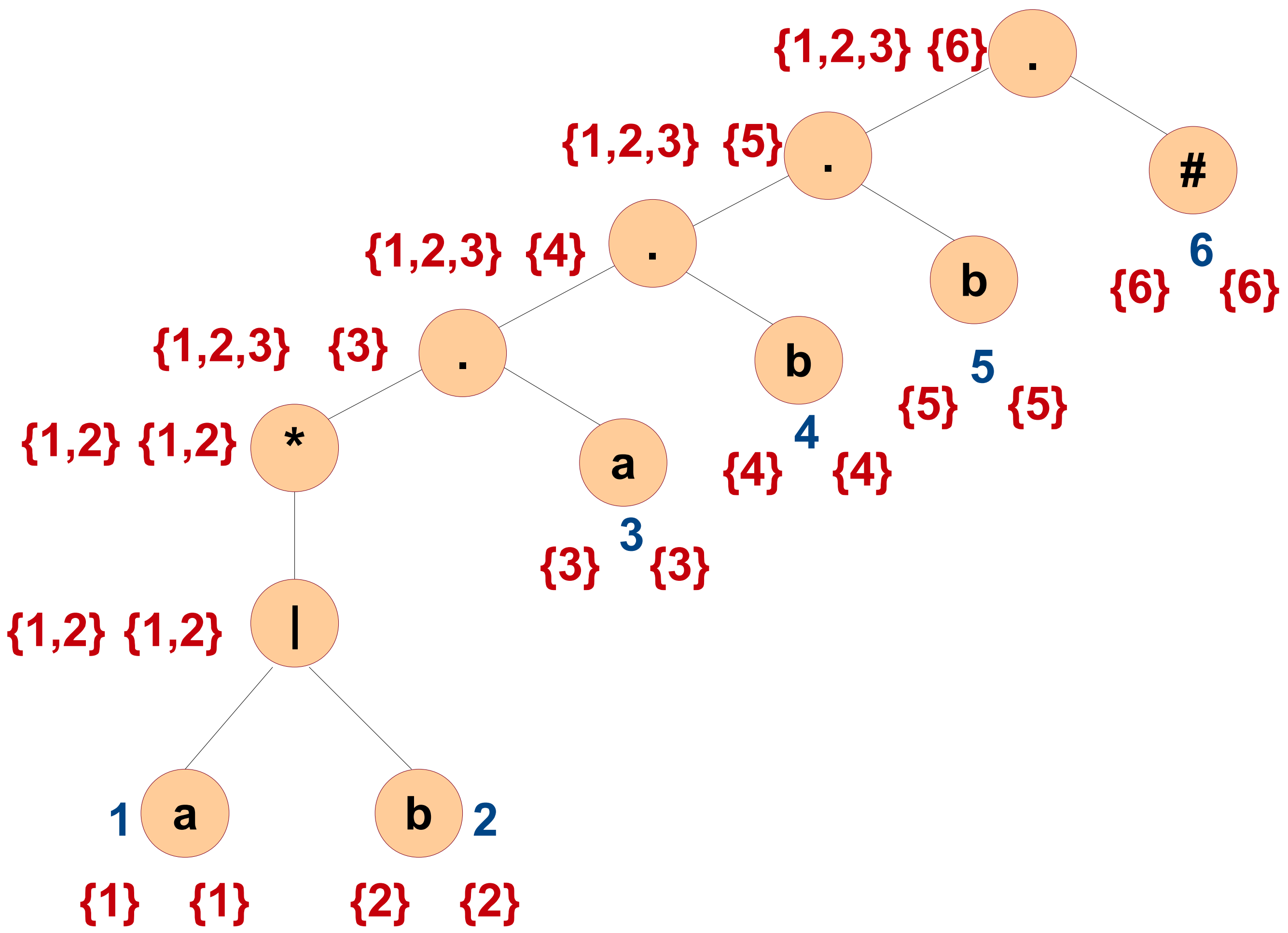
lastpos

Node n	**lastpos(n)**
leaf labeled ϵ	{ }
leaf with position i	{i}
or-node n = c1 | c2	lastpos(c1) U lastpos(c2)

cat-node n = c1c2	if (nullable(c2)) lastpos(c1) U lastpos(c2)
	else lastpos(c2)

star-node n = c*	lastpos(c)

firstpos lastpos

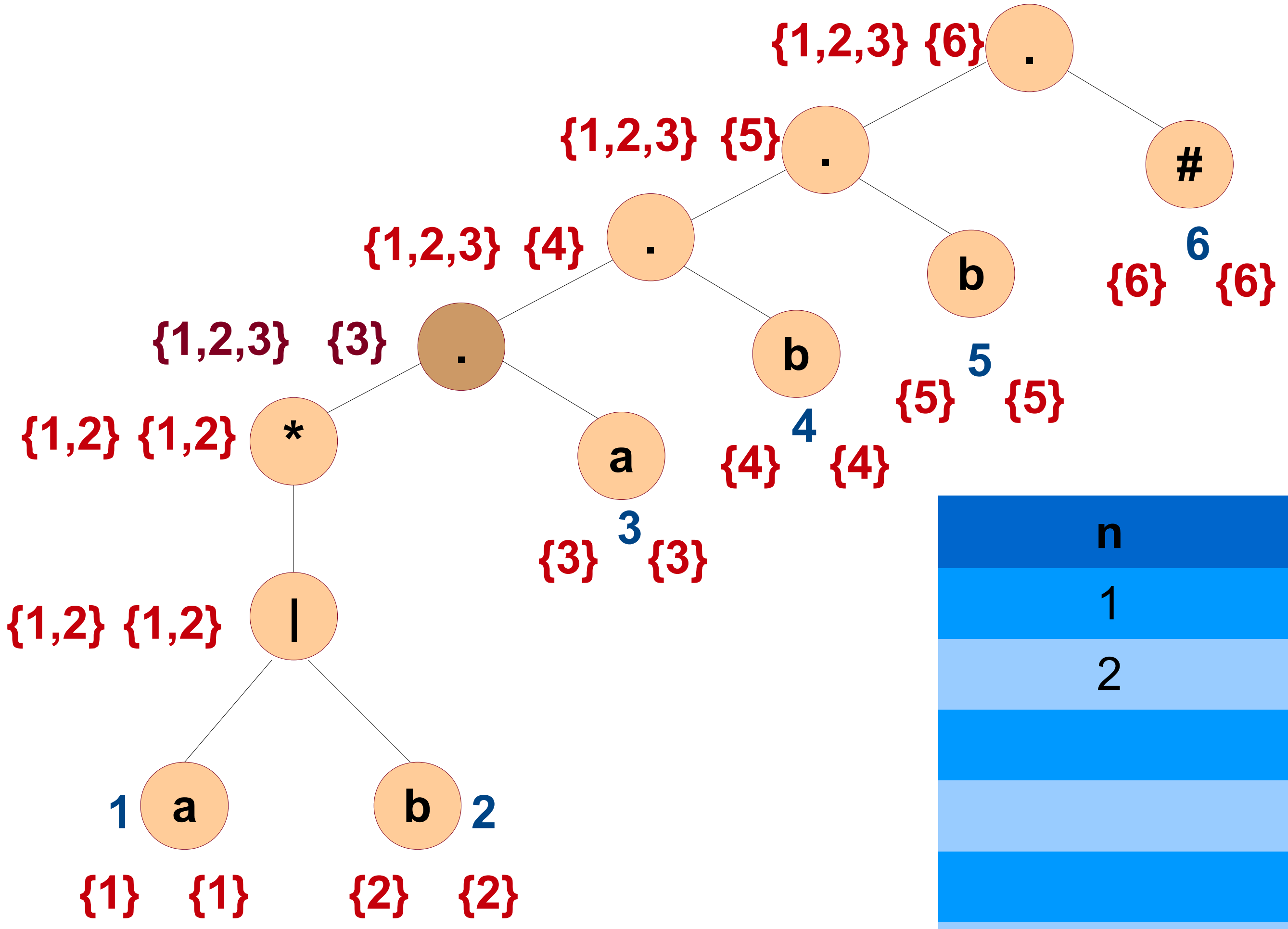


followpos

- *followpos*(n): set of next possible positions from n for valid strings.
  - If n is a **cat-node** with child nodes c1 and c2, then for each position in *lastpos(c1)*, all positions in *firstpos(c2)* *follow*.
  - If n is a **star-node**, then for each position in *lastpos(n)*, all positions in *firstpos(n)* *follow*.

followpos

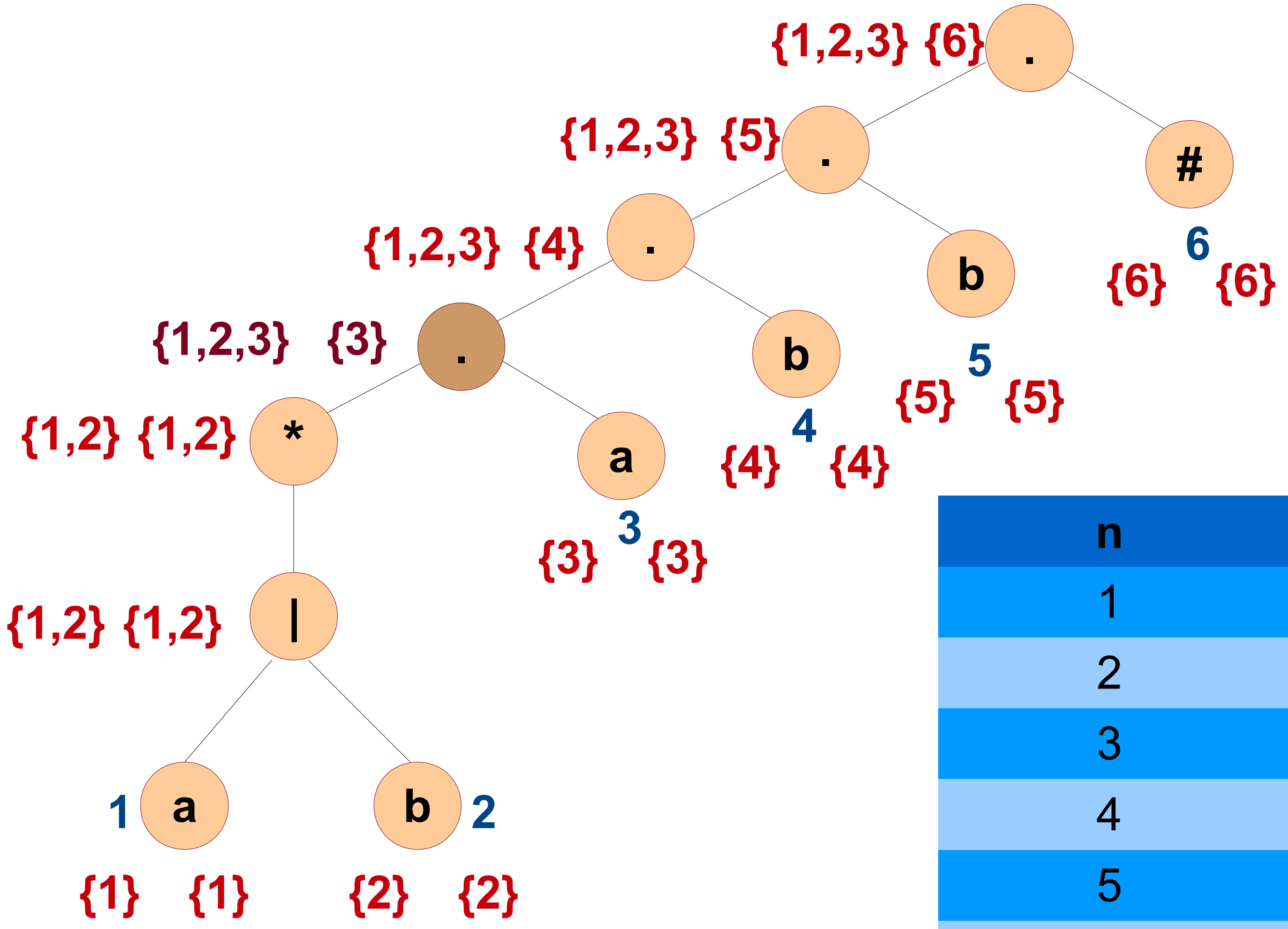
If n is a **cat-node** with child nodes c1 and c2, then for each position in *lastpos(c1)*, all positions in *firstpos(c2)* *follow*.



n	followpos(n)
1	{3}
2	{3}

followpos

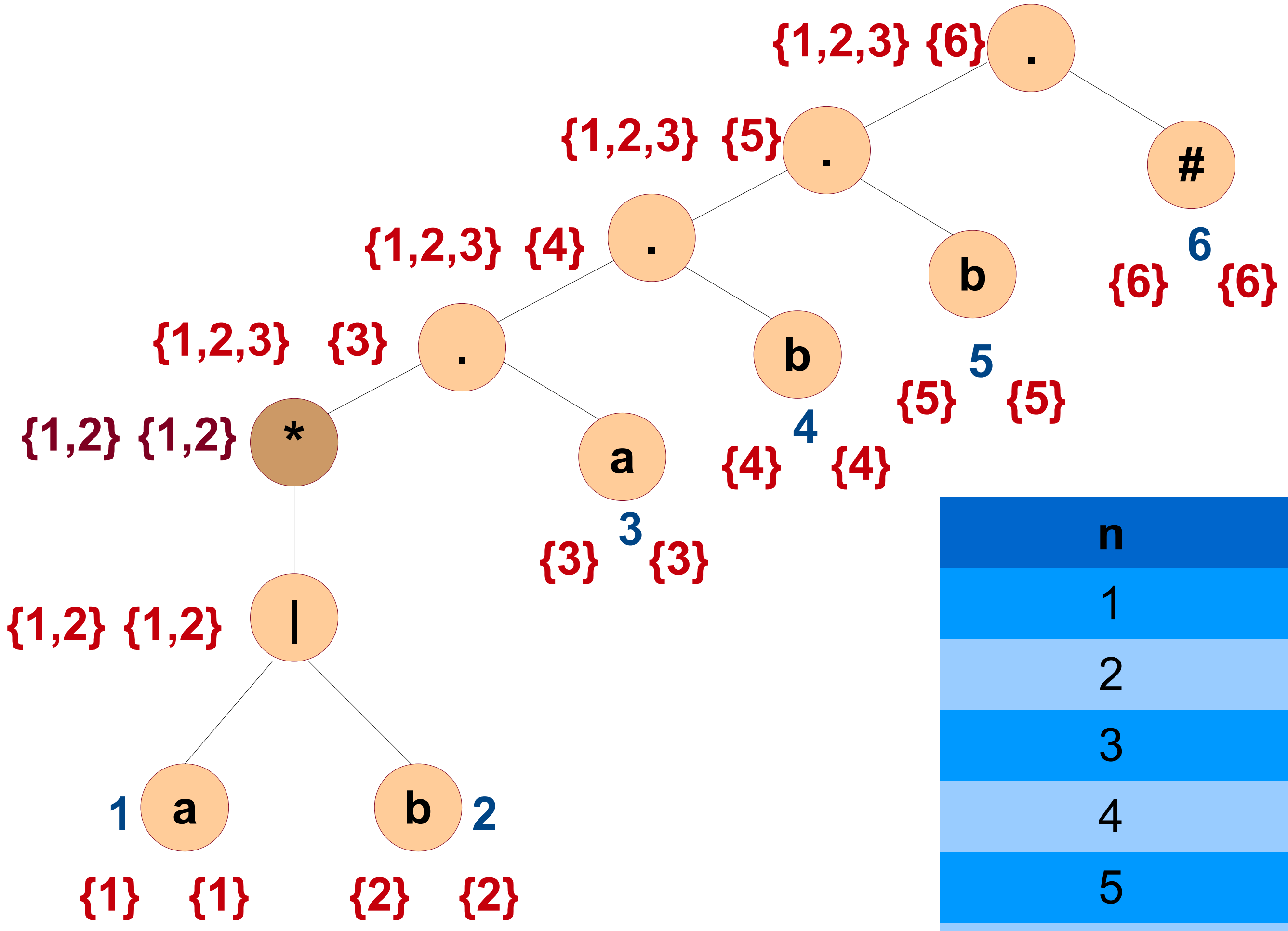
If n is a **cat-node** with child nodes c1 and c2, then for each position in *lastpos(c1)*, all positions in *firstpos(c2)* *follow*.



n	followpos(n)
1	{3}
2	{3}
3	{4}
4	{5}
5	{6}
6	{ }

followpos

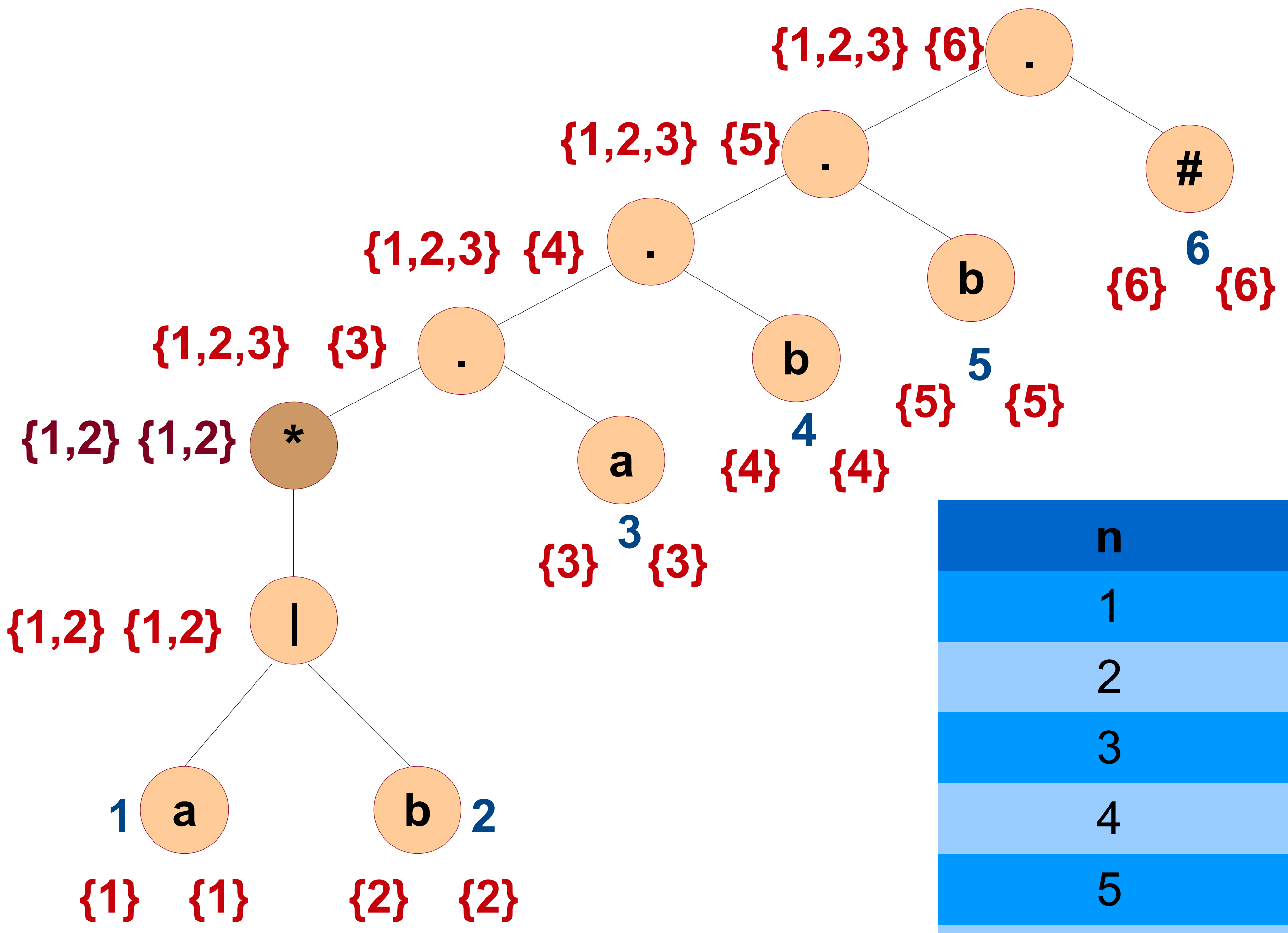
If n is a **star-node**, then for each position in *lastpos(n)*, all positions in *firstpos(n)* *follow*.



n	followpos(n)
1	{3}
2	{3}
3	{4}
4	{5}
5	{6}
6	{ }

followpos

If n is a **star-node**, then for each position in *lastpos(n)*, all positions in *firstpos(n)* *follow*.



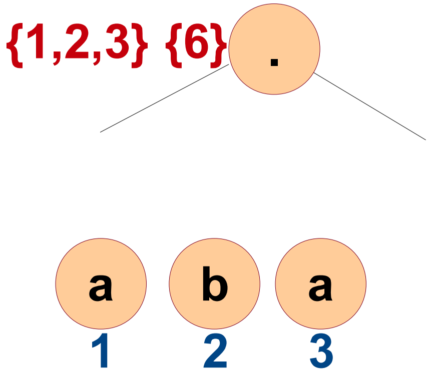
n	followpos(n)
1	{3, 1, 2}
2	{3, 1, 2}
3	{4}
4	{5}
5	{6}
6	{ }

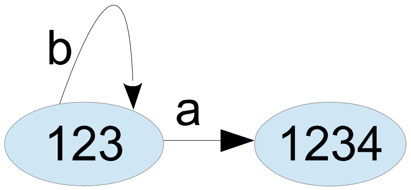
Regex → DFA

1. Construct a syntax tree for regex#.
2. Compute *nullable*, *firstpos*, *lastpos*, *followpos*.
3. Construct DFA using transition function *(next slide)*.
4. Mark *firstpos(root)* as start state.
5. Mark states that contain position of # as accepting states.

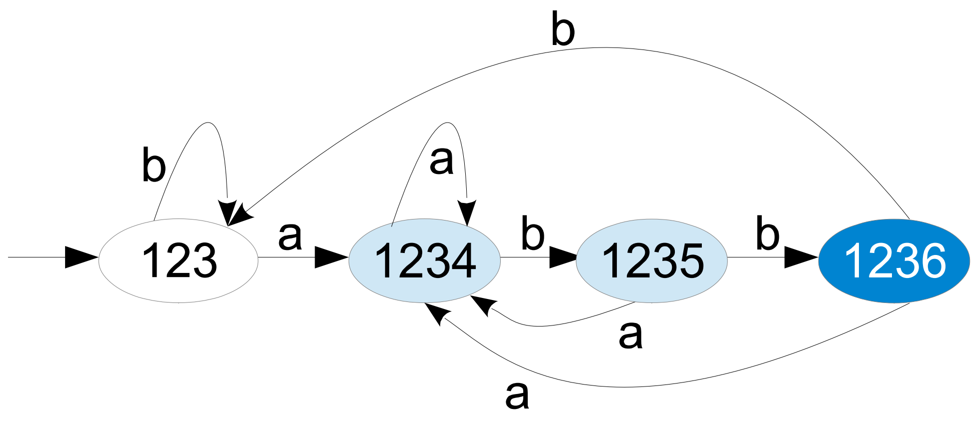
DFA Transitions

```
create unmarked state firstpos(root).
while there exists unmarked state s {
    mark s
    for each input symbol a {
        uf = U followpos(p) where p is in s labeled a
        transition[s, a] = uf
        if uf does not exist
            unmark uf
    }
}
```

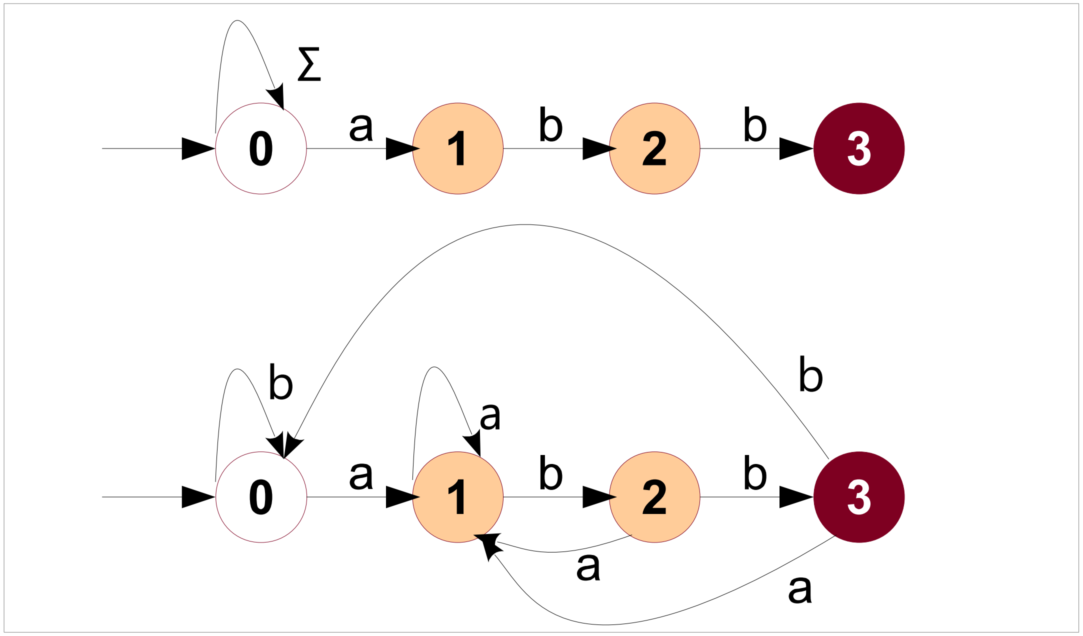




Final DFA



DFA



NFA

DFA

In case you are wondering...

- What to do with this DFA?
  - Recognize strings during lexical analysis.
  - Could be used in utilities such as *grep*.
  - Could be used in regex libraries as supported in php, python, perl, ... and Vipin's Ruby.