Regular Expression, NFA, DFA, Minimization of DFA

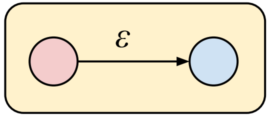
Md Shad Akhtar
Assistant Professor
IIIT Dharwad

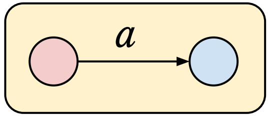
Regular Expression to DFA

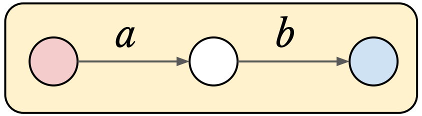
- Approaches to systematically convert a regular expression to an equivalent DFA.
  - Using Thompson construction
    - RE → ε-NFA → DFA
  - Using Syntax Tree
    - RE → DFA
  - Etc.

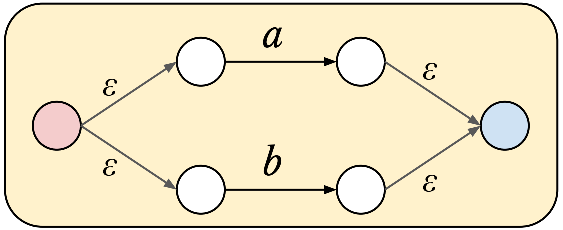
Thompson Construction for RE → ε-NFA

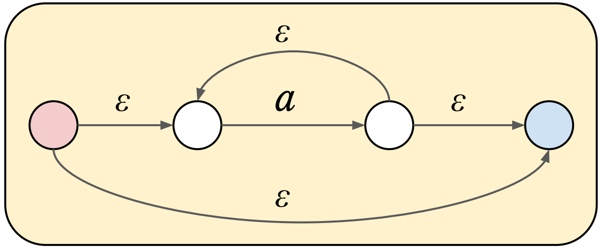
- It guarantees that the resulting NFA will have exactly one final state, and one start state
- Define an ε-NFA construct for each basic regular expression (ε, a, ab, a+b, a*)
- Combine multiple basic ε-NFA constructs to obtain ε-NFA for more complex regular expressions.











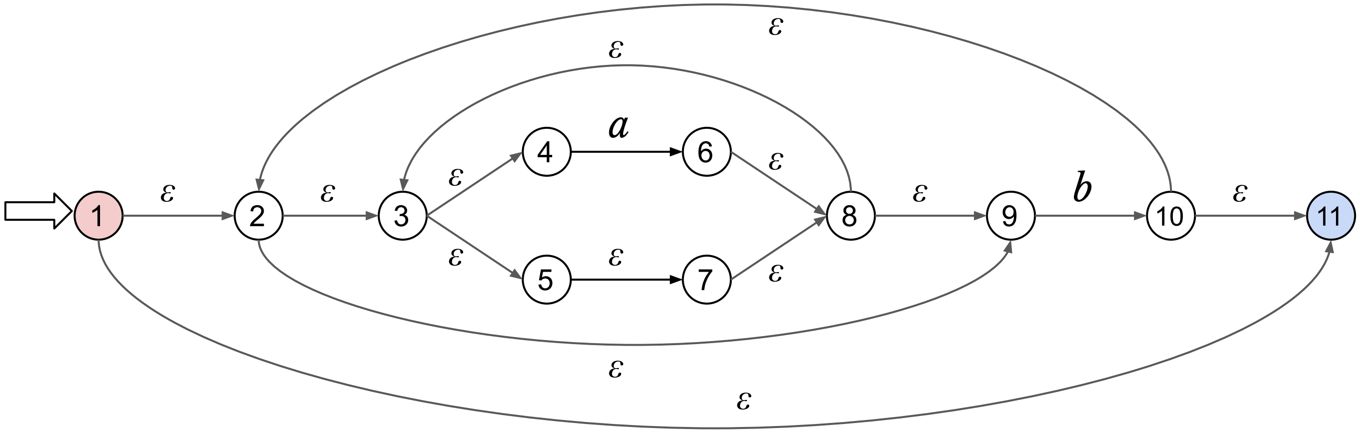
Red state: Start Blue state: Final

RE to ε-NFA for ((ε+a)*b)*

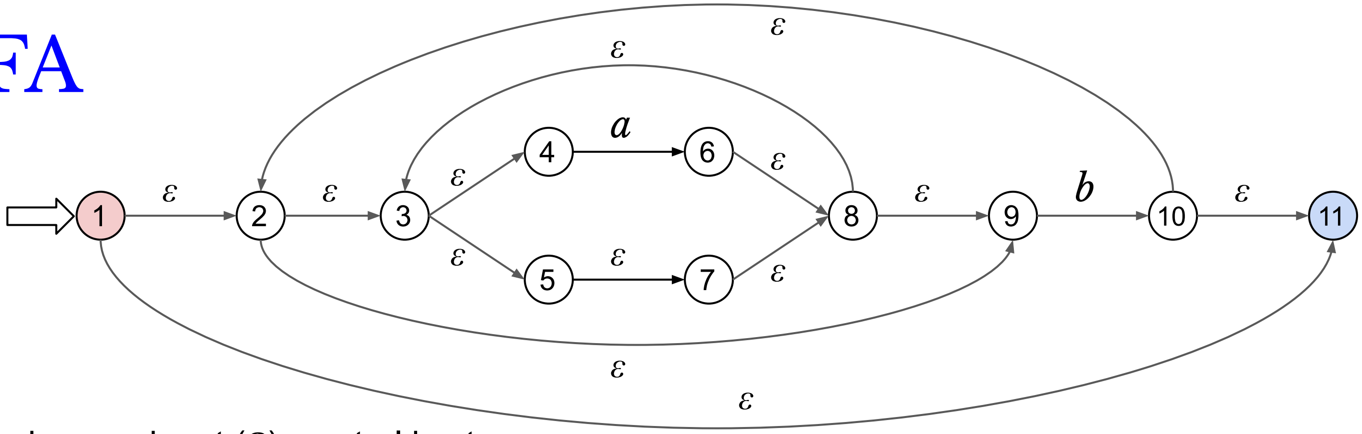
Thompson construction

- ε
- a
- ε + a
- (ε + a)*
- (ε + a)*b
- ((ε + a)*b)*

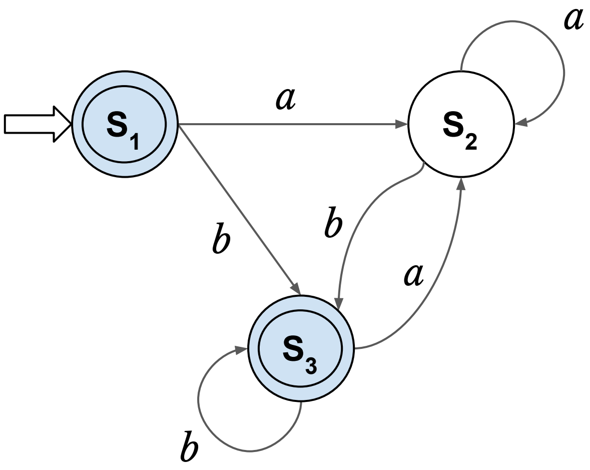
Number the states



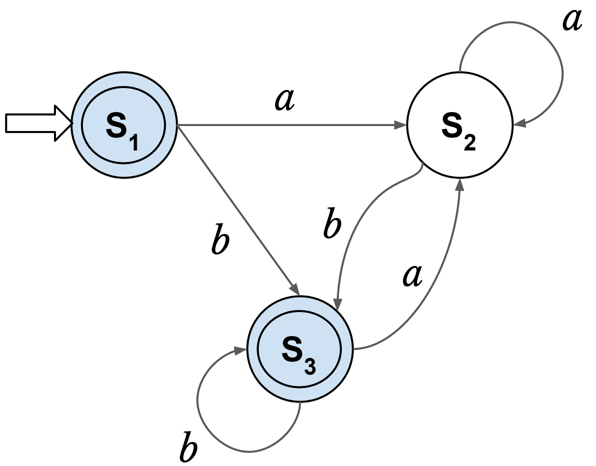
ε-NFA to DFA



- At the start, without consuming any input (Ɛ), control is at the following states
  - {1,2,3,4,5,7,8,9,11} → $\mathbf{S_1}$
- S_1 on **a**,
  - {6,8,9,3,4, 5,7} → $\mathbf{S_2}$
- S_1 on **b**,
  - {10,11,2,3,4,5,7,8,9} → $\mathbf{S_3}$
- S_2 on **a**,
  - {6,8,9,3,4, 5,7} → $\mathbf{S_2}$
- S_2 on **b**,
  - {10,11,2,3,4,5,7,8,9} → $\mathbf{S_3}$
- S_3 on **a**,
  - {6,8,9,3,4, 5,7} → $\mathbf{S_2}$
- S_3 on **b**,
  - {10,11,2,3,4,5,7,8,9} → $\mathbf{S_3}$



Minimal DFA



- Divide the states into two sets
  - Accepting states and Non-accepting states

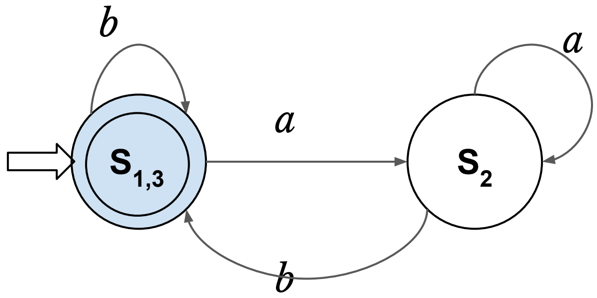
- Try to split the sets into disjoint subset, only if
  - On any input symbols their transitions is different

- All the states with same transition on all the input symbols remains together.

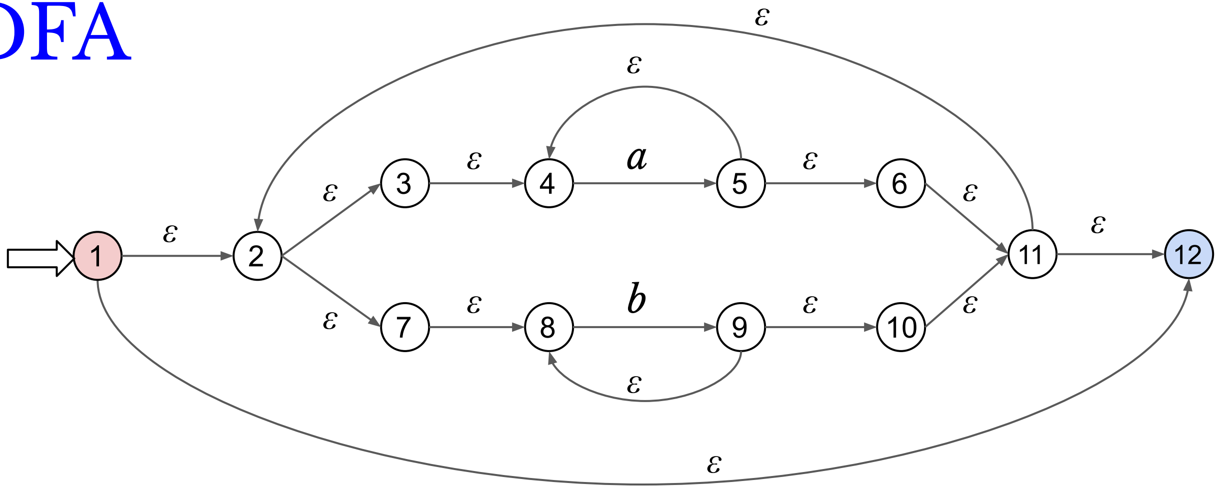
- Accepting states = {$\mathbf{S_1, S_3}$}
- Non-accepting states = {$\mathbf{S_2}$}
- Non-accepting set has one element, thus it cannot be splitted
- For accepting set,
  - $\mathbf{S_1}$ on **a** $\Rightarrow$ $\mathbf{S_2}$
  - $\mathbf{S_3}$ on **a** $\Rightarrow$ $\mathbf{S_2}$

  - $\mathbf{S_1}$ on **b** $\Rightarrow$ $\mathbf{S_3}$
  - $\mathbf{S_3}$ on **b** $\Rightarrow$ $\mathbf{S_3}$

Transitions are same, so cannot be splitted



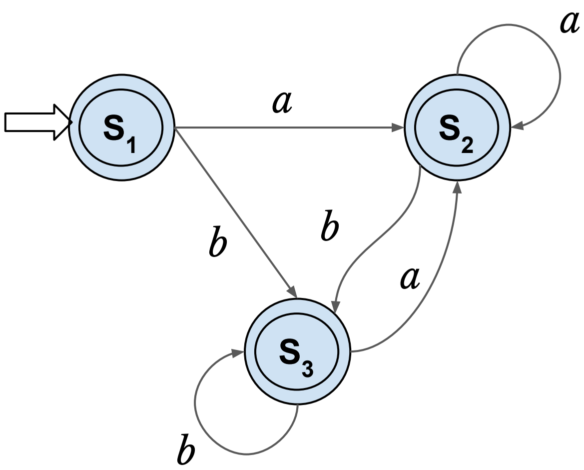
RE to ε-NFA to DFA

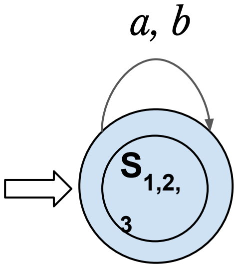


(aa* + bb*)*

- (aa* + bb*)* ⇒ $(a^+ + b^+)^*$

- Start
  - {1,2,3,4,7,8,12} ⇒ S_1
- S_1 on a
  - {5,6,11,12,4,2,3,7,8} ⇒ S_2
- S_1 on b
  - {9,10,11,12,8,2,3,7,4 ⇒ S_3}
- S_2 on a
  - {5,6,11,12,4,2,3,7,8} ⇒ S_2
- S_2 on b
  - {9,10,11,12,8,2,3,7,4 ⇒ S_3}
- S_3 on a
  - {5,6,11,12,4,2,3,7,8} ⇒ S_2
- S_3 on b
  - {9,10,11,12,8,2,3,7,4 ⇒ S_3}



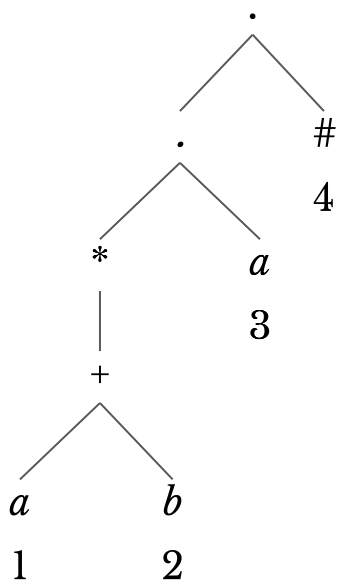


Syntax Tree approach for RE → DFA

- Regular expression can be directly converted into a DFA (without creating a ε-NFA first)
- Augment the given regular expression by concatenating it with a special symbol #
  - r → (r)# augmented regular expression
- Create a syntax tree for this augmented regular expression
- Syntax tree
  - Leaves: alphabet symbols (including # and the empty string) in the augmented regular expression
  - Intermediate nodes: operators
- Compute four functions on syntax tree: *followpos*, *firstpos*, *lastpos* and *nullable*

RE → DFA for *(a+b)*a*

- Augmented RE
  - $(a+b)^* a \rightarrow (a+b)^* a \#$
- Syntax tree
  - Each symbol is at a leave
  - Inner nodes are operators
  - Number each symbol



Functions *firstpos*, *lastpos*, and *nullable*

- firstpos(n)
  - Set of the positions of the first symbols of strings generated by the sub-expression rooted by n

- *lastpos(n)*
  - 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

Functions *firstpos*, *lastpos*, and *nullable*

n	***nullable*(*n*)**	***firstpos*(*n*)**	***lastpos*(*n*)**
Leaf labelled as ε	True	ϕ	ϕ
Leaf labelled with position *i*	False	{i}	{i}
+ c1 c2	*nullable*(c1) Or *nullable*(c2)	*firstpos*(c1) ∪ *firstpos*(c2)	*lastpos*(c1) ∪ *lastpos*(c2)
. c1 c2	*nullable*(c1) And *nullable*(c2)	If *nullable*(c1) *firstpos*(c1) ∪ *firstpos*(c2) else *firstpos*(c1)	If *nullable*(c2) *lastpos*(c1) ∪ *lastpos*(c2) else *lastpos*(c2)
* c1	True	*firstpos*(c1)	*lastpos*(c1)

Functions *firstpos*, *lastpos*, and *nullable* for *(a+b)*a#*

n	*nullable*(*n*)	*firstpos*(*n*)	*lastpos*(*n*)
1	False	{1}	{1}
2	False	{2}	{2}
3	False	{3}	{3}
4	False	{4}	{4}
+ 1 2	False	{1,2}	{1,2}
* | +	True	{1,2}	{1,2}

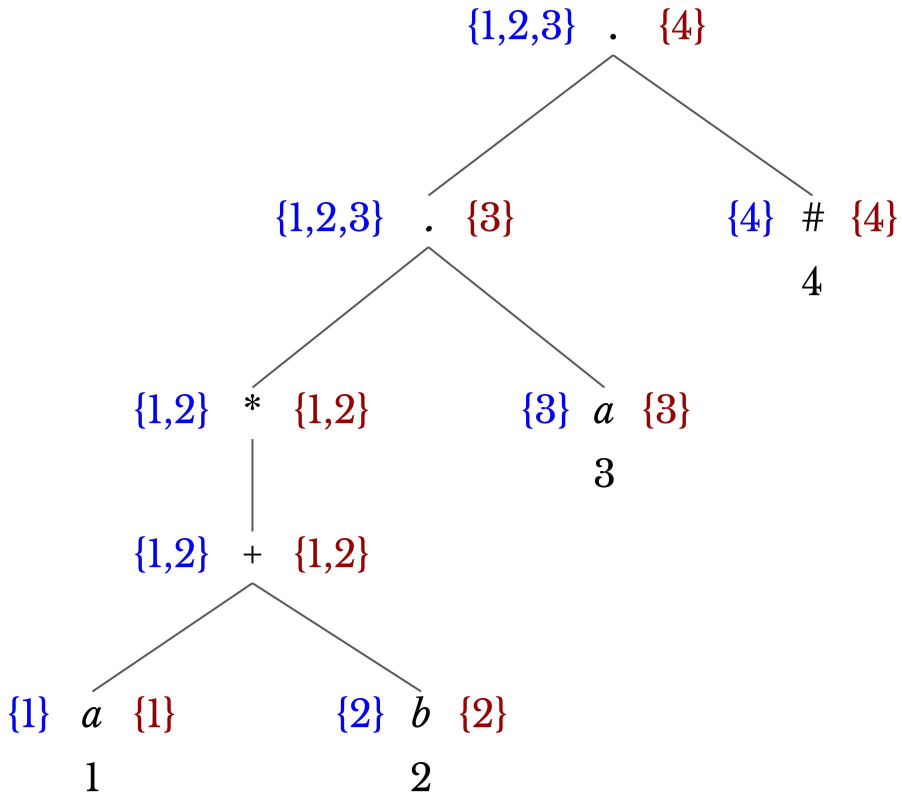
Functions *firstpos*, *lastpos*, and *nullable* for *(a+b)*a#*

n	*nullable*(*n*)	*firstpos*(*n*)	*lastpos*(*n*)
3	False	{3}	{3}
4	False	{4}	{4}
· * 3	False	{1,2,3}	{3}
· · 4	False	{1,2,3}	{4}

Annotated syntax tree for

Blue → firstpos(n)

Red → lastpos(n)



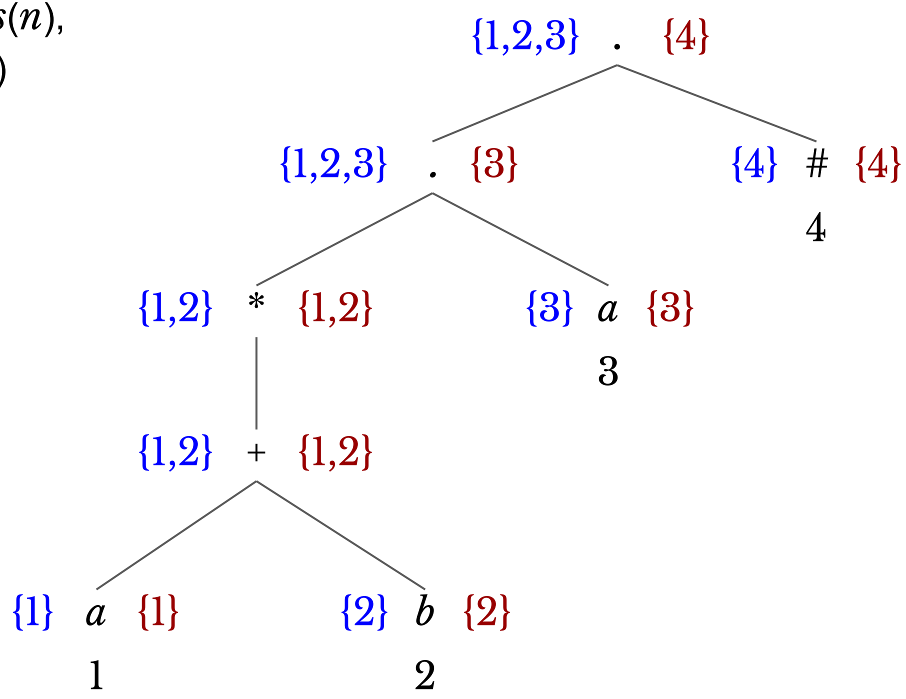
Function *followpos*

- Define the function *followpos* for each and only leaf positions
- $followpos(i)$
    - Set of positions which can follow the position i in the strings generated by the augmented regular expression

Computing *followpos*(i)

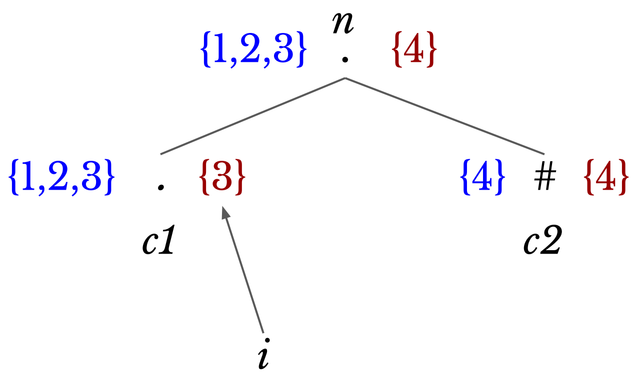
- If n is concatenation-node with left child c1 and right child c2, and i is a position in *lastpos*(c1),
  a. *followpos*(i) ← *followpos*(i) U *firstpos*(c2)
- If n is a star-node, and i is a position in *lastpos*(n),
  a. *followpos*(i) ← *followpos*(i) U *firstpos*(n)

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



Computing *followpos*(i)

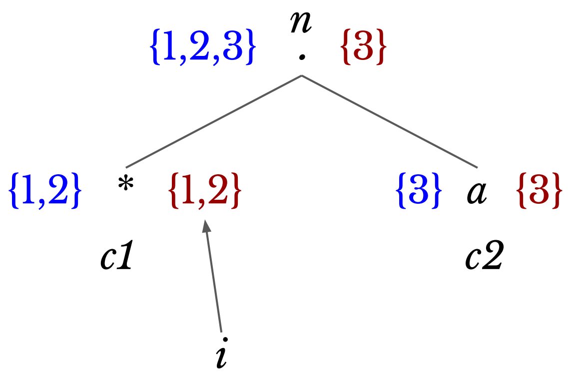
- If n is concatenation-node with left child c1 and right child c2, and i is a position in *lastpos*(c1),
  a. *followpos*(i) ← *followpos*(i) U *firstpos*(c2)
- If n is a star-node, and i is a position in *lastpos*(n),
  a. *followpos*(i) ← *followpos*(i) U *firstpos*(n)



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

Computing *followpos*(i)

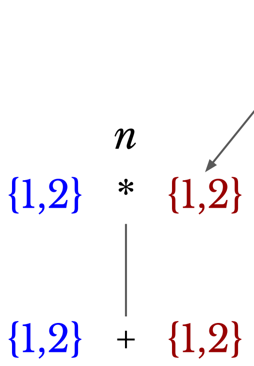
- If n is concatenation-node with left child c1 and right child c2, and i is a position in *lastpos*(c1),
  a. *followpos*(i) ← *followpos*(i) U *firstpos*(c2)
- If n is a star-node, and i is a position in *lastpos*(n),
  a. *followpos*(i) ← *followpos*(i) U *firstpos*(n)



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

Computing *followpos*(i)

- If n is concatenation-node with left child c1 and right child c2, and i is a position in *lastpos*(c1),
  a. $followpos(i) \leftarrow followpos(i) \cup firstpos(c2)$
- If n is a star-node, and i is a position in $lastpos(n)$,
  a. $followpos(i) \leftarrow followpos(i) \cup firstpos(n)$

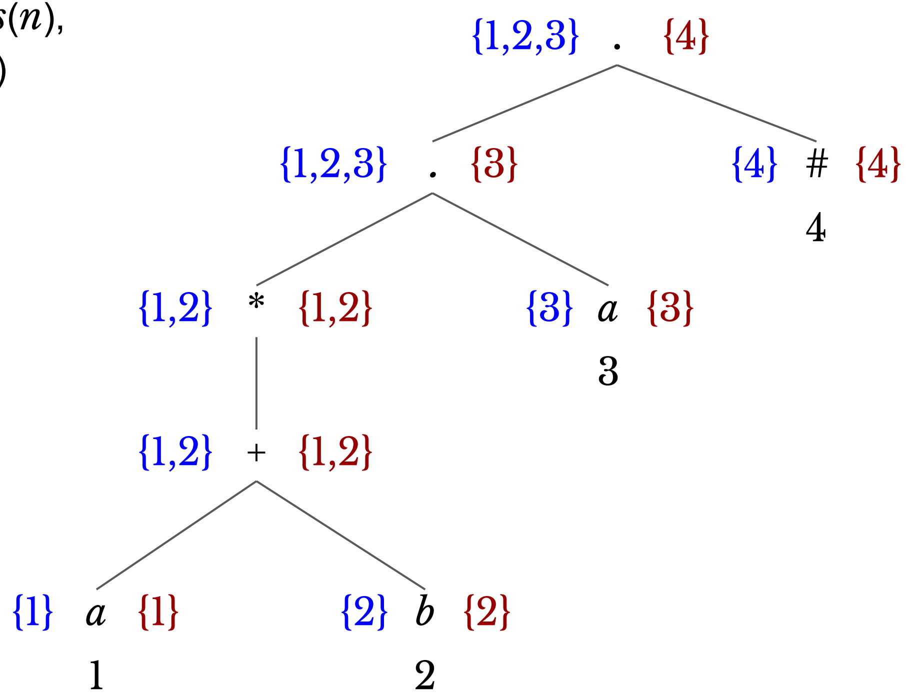


$followpos(1) = \{1,2,3\}$
$followpos(2) = \{1,2,3\}$
$followpos(3) = \{4\}$
$followpos(4) = \{\}$

Computing *followpos(i)*

- If n is concatenation-node with left child c1 and right child c2, and i is a position in *lastpos*(c1),
  a. *followpos*(i) ← *followpos*(i) U *firstpos*(c2)
- If n is a star-node, and i is a position in *lastpos*(n),
  a. *followpos*(i) ← *followpos*(i) U *firstpos*(n)

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



Complete Algorithm (RE→ DFA)

1. Create the syntax tree of (r) #
2. Calculate the functions: *followpos*, *firstpos*, *lastpos*, *nullable*
3. Put *firstpos*(root) into the states of DFA as an unmarked state
4. **while** (there is an unmarked state *S* in the states of DFA) **do**
    a. mark *S*
    b. **for each** input symbol *a* **do**
        i. let $s_1, .., s_n$ are positions in *S* and symbols in those positions are *a*
    c. $S' \leftarrow$ *followpos*(s_1) ∪ ... ∪ *followpos*(s_n)
    d. move(*S*, *a*) $\leftarrow S'$
    e. **if** (*S'* **is not** empty **and not** in the states of DFA)
        i. put *S'* into the states of DFA as an unmarked state
5. the start state of DFA is *firstpos*(root)
6. the accepting states of DFA are all states containing the position of #

$\Sigma = \{a, b\}$

$(a + b)^* \; a \; \#$
1 2 3 4

RE→ DFA for *(a+b)*a*

1. DFA states = $\{S^0_{\{1,2,3\}}\}$
2. DFA states = $\{\cancel{S}^0_{\{1,2,3\}}\}$
3. $S^0_{\{1,2,3\}}$ on a
   a. Positions in S with symbol a = {1, 3}
   b. S = *followpos*{1} U *followpos*{3} = {1,2,3,4} = S^1
4. DFA states = $\{\cancel{S}^0_{\{1,2,3\},}\ ,\ S^1_{\{1,2,3,4\}}\}$
5. $S^0_{\{1,2,3\}}$ on b
   a. Positions in S with symbol b = {2}
   b. S = *followpos*{2} = {1,2,3} = S^0
6. DFA states = $\{\cancel{S}^0_{\{1,2,3\},}\ ,\ \cancel{S}^1_{\{1,2,3,4\}}\}$
7. $S^1_{\{1,2,3,\ 4\}}$ on a
   a. Positions in S with symbol a = {1, 3}
   b. S = *followpos*{1} U *followpos*{3} = {1,2,3,4} = S^1
8. $S^1_{\{1,2,3\}}$ on b
   a. Positions in S with symbol b = {2}
   b. S = *followpos*{2} = {1,2,3} = S^0