Principles of Programming Languages

http://www.di.unipi.it/~andrea/Didattica/PLP-15/

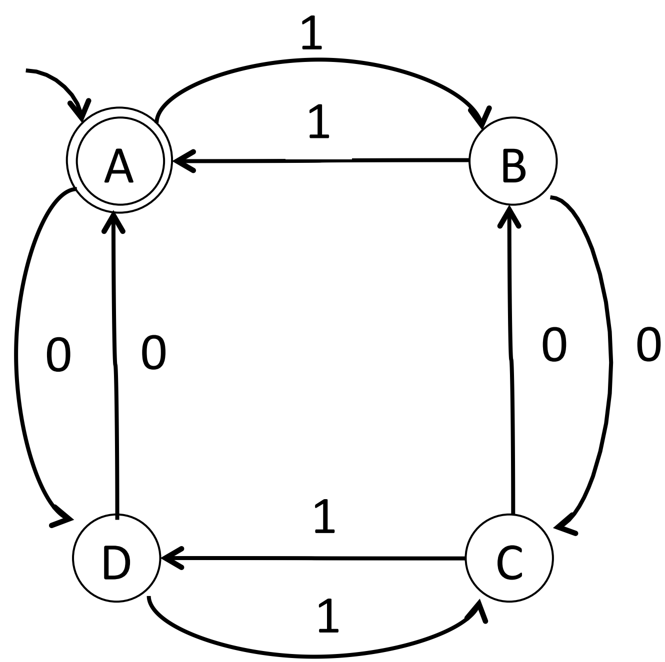
Prof. Andrea Corradini
Department of Computer Science, Pisa

Lesson 7

- From DSA to Regular Expression
- From Regular Expressions to DSA, directly

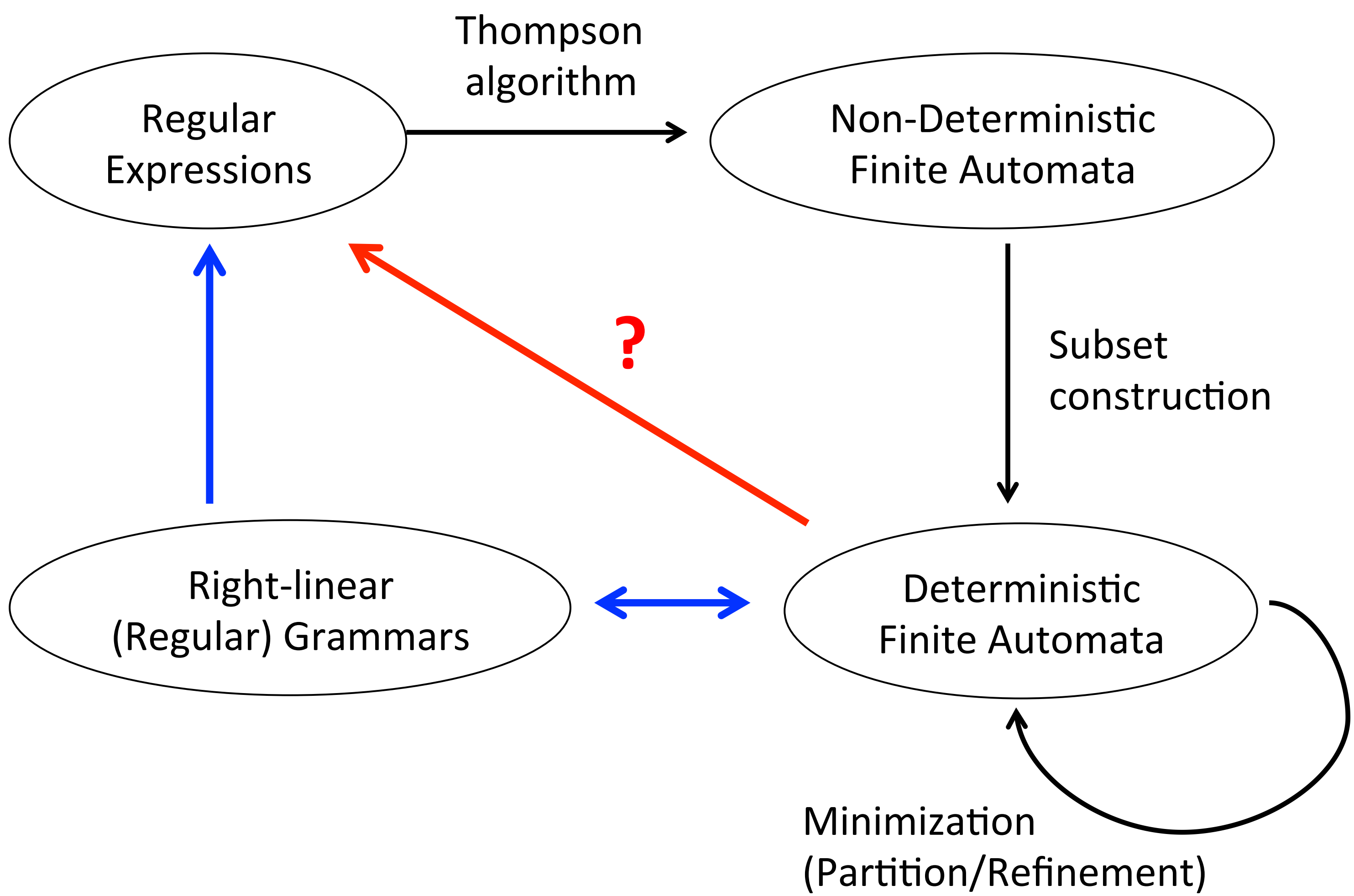
Motivations: yesterday's exercise 7(b)

- Write a regular expression over the set of symbols {0,1} that describes the language of all strings having an even number of 0's and of 1's
  - Not easy….
  - A solution: (00|11)*((01|10)(00|11)*(01|10)(00|11)*)*
  - How can we get it?



- Towards the solution: a deterministic automaton accepting the language
- But how do we get the regular expression defining the language accepted by the automaton?

Regular expressions, Automata, and all that…

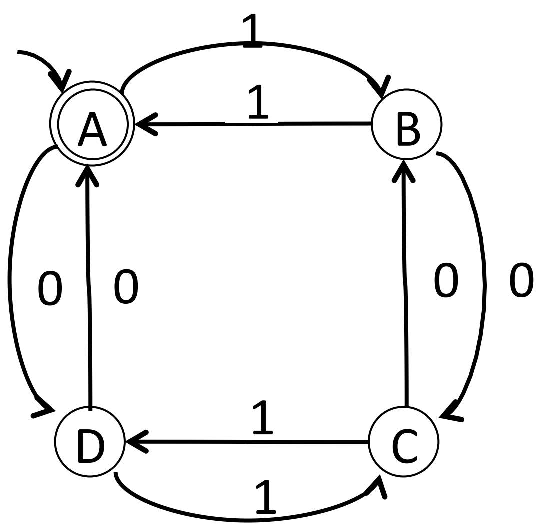


From automata to Regular Expressions

- Three approaches:
  - Dynamic Programming [Scott, Section 2.4 on CD] [Hopcroft, Motwani, Ullman, *Introduction to Automata Theory, Languages and Computation*, Section 3.2.1]
  - Incremental state elimination [HMU, Section 3.2.2]
  - Regular Expression as fixed-point of a continuous function on languages

DFAs and Right-linear Grammars

- In a *right-linear* (*regular*) grammar each production is of the form $A \to w\,B$ or $A \to w$ $(w \in T^*)$
- From a DFA to a right-linear grammar



$A \to \varepsilon \mid 1B \mid 0D$
$B \to 1A \mid 0C$
$C \to 0B \mid 1D$
$D \to 0A \mid 1C$

- The construction also works for NFA
- A similar construction can transform any right-linear grammar into an NFA (productions might need to be transformed introducing new non-terminals)

Kleene fixed-point theorem

- A *complete partial order (CPO)* is a partial order with a least element $\perp$ and such that every increasing chain has a supremum
- Theorem: *Every continuous function F over a complete partial order (CPO) has a least fixed-point, which is the supremum of chain*

$$F(\perp) \leq F(F((\perp)) \leq \ldots \leq F^n(\perp) \leq ..$$

Context Free grammars as functions on the CPO of languages

- Languages over Σ form a *complete partial order* under set inclusion
- A context free grammar defines a continuous function over (tuples of) languages
  - **A -> *a* | *b*A** $F(L) = \{a\} \cup \{bw \mid w \in L\}$
- The language generated by the grammar is the least-fixed point of the associated function
  - $\varnothing \subset \{a\} \subset \{a, ba\} \subset \{a, ba, bba\} \subset \ldots \subset \{b^n a \mid n \geq 0\}$
- In the case of right-linear grammars we can describe the least fixed-point as a regular expression
  - *Lang(***A***) = b*a*

Example: from right-linear grammar to regular expression

A → ε | 1B | 0D
B → 1A | 0C
C → 0B | 1D
D → 0A | 1C

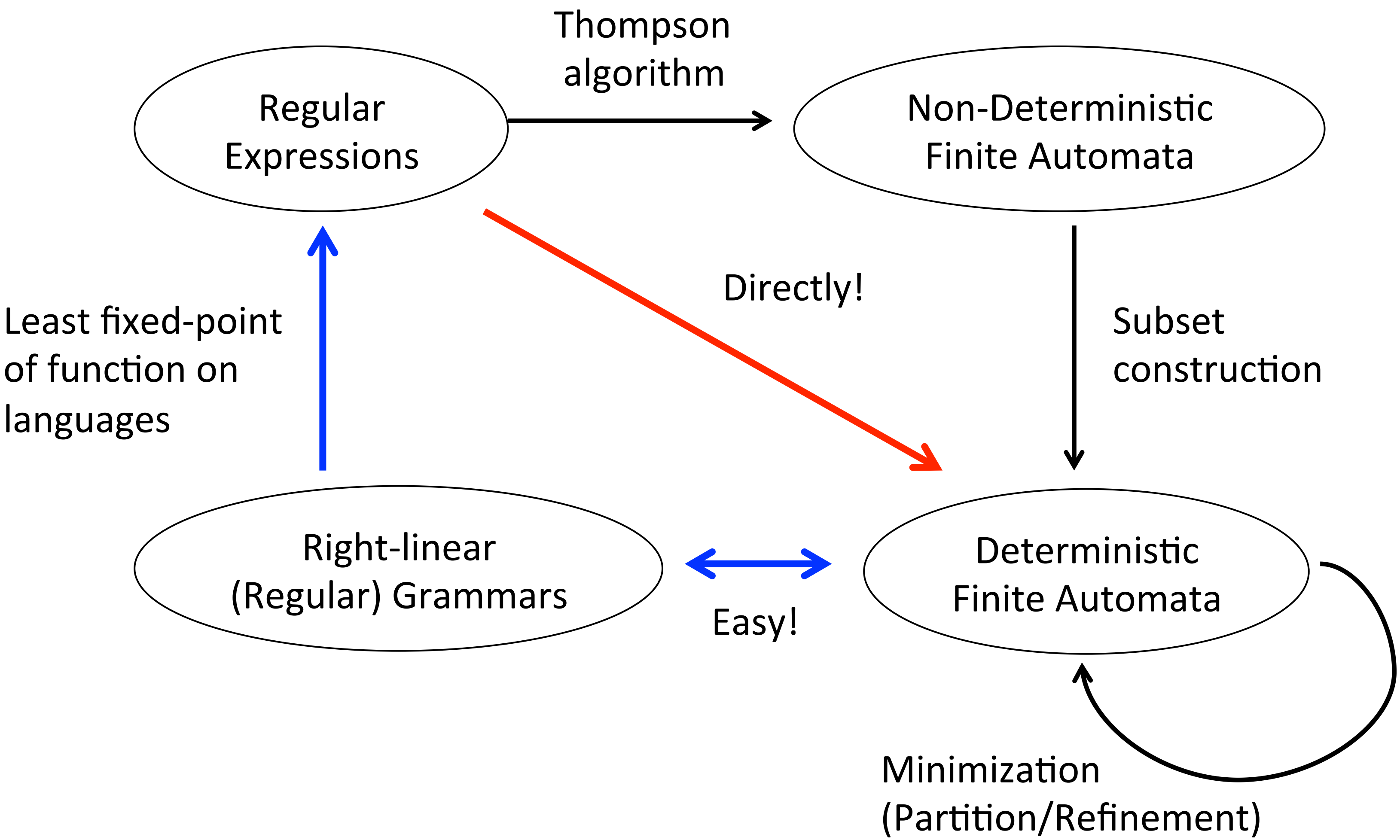
1) Substitute D in A and C
A → ε | 1B | 0(0A | 1C)
B → 1A | 0C
C → 0B | 1(0A | 1C)

2) Substitute B in A and C
A → ε | 1(1A | 0C) | 0(0A | 1C)
C → 0(1A | 0C) | 1(0A | 1C)

3) Put C in form C = α | βC
A → ε | 1(1A | 0C) | 0(0A | 1C)
C → 01A | 10A | (00 | 11)C
4) Solve C: C = (00 | 11)*(01A | 10A)

5) Factorize C in A
A → ε | 11A | 00A | (10 | 01)C
6) Substitute C in A
A → ε | 11A | 00A | (10 | 01) (00 | 11)*(01A | 10A)
7) Put A in form A = α | βA
A → ε | (11 | 00 | (10 | 01) (00 | 11)*(01 | 10))A
8) Solve A: A = (11 | 00 | (10 | 01) (00 | 11)*(01 | 10))*
The other solution: (00|11)*((01|10)(00|11)*(01|10)(00|11)*)*

Regular expressions, Automata, and all that…

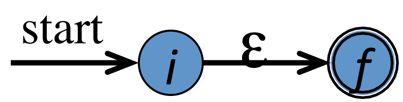


From Regular Expression to DFA Directly

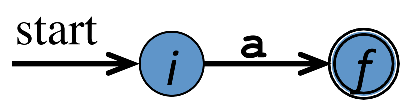
- The "*important states*" of an NFA are those with a non-ε outgoing transition,
  - if $move(\{s\}, a) \neq \emptyset$ for some a then s is an important state
- The subset construction algorithm uses only the important states when it determines $\varepsilon\text{-}closure(move(T, a))$

What are the “important states” in the NFA built from Regular Expression?

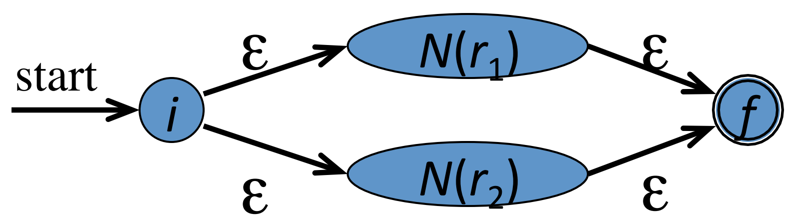
ε



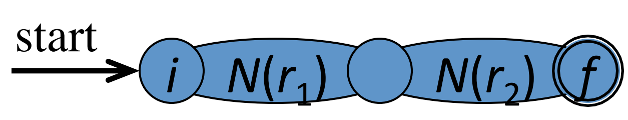
a



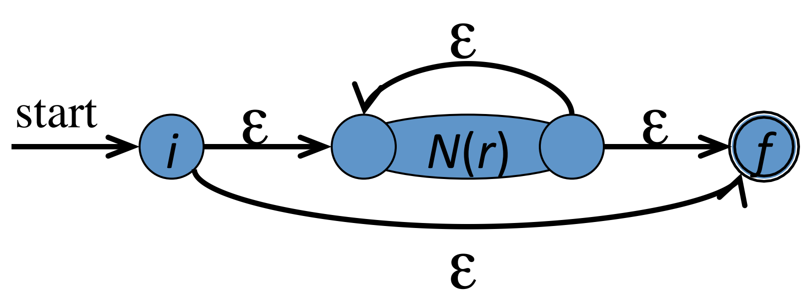
$r_1 \mid r_2$



$r_1 r_2$



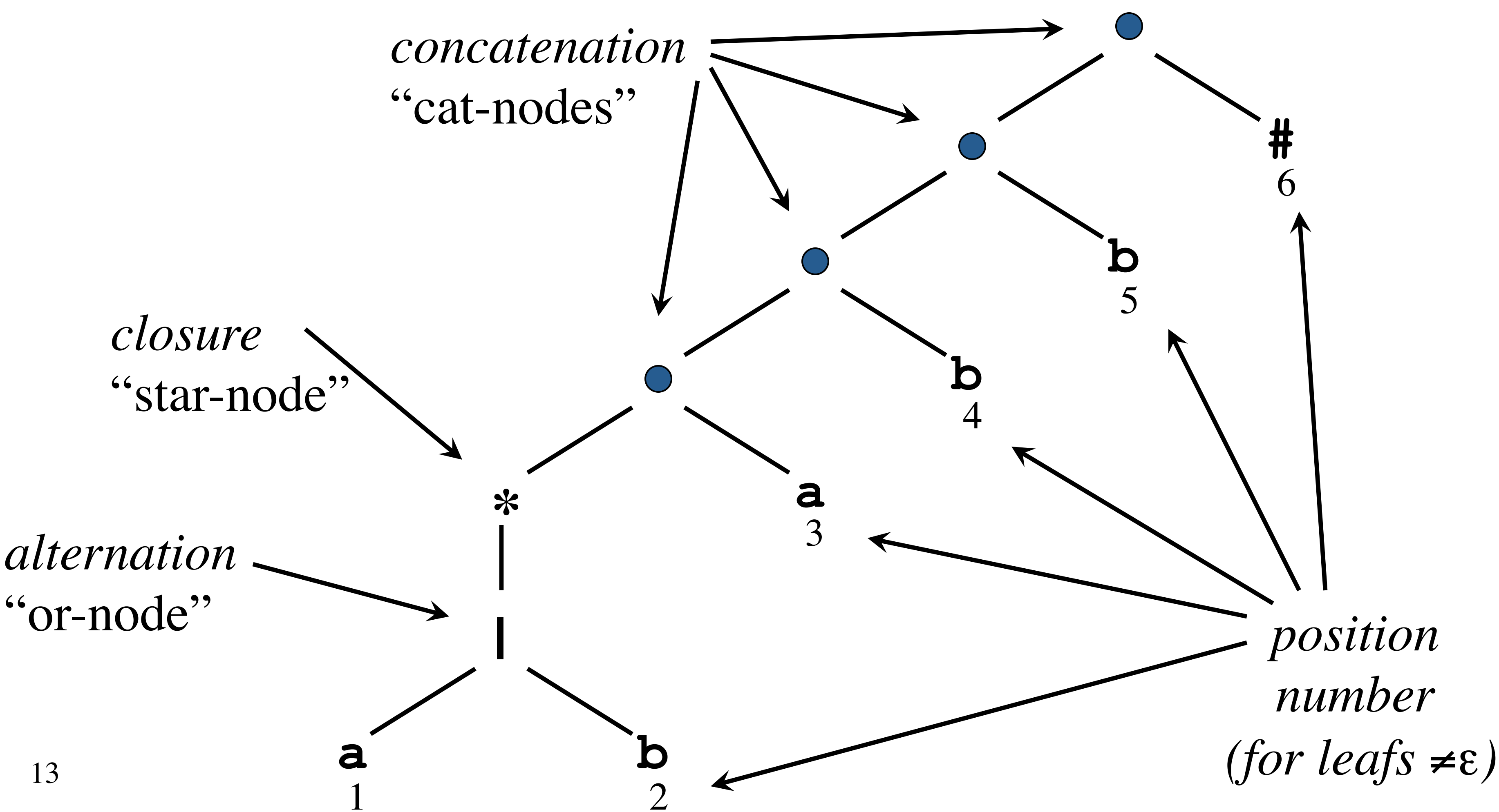
r^*



From Regular Expression to DFA Directly (Algorithm)

- The only accepting state (via the Thompson algorithm) is not important
- Augment the regular expression *r* with a special end symbol # to make accepting states important: the new expression is *r#*
- Construct a syntax tree for *r#*
- Attach a unique integer to each node not labeled by ε

From Regular Expression to DFA Directly: Syntax Tree of **`(a|b)*abb#`**



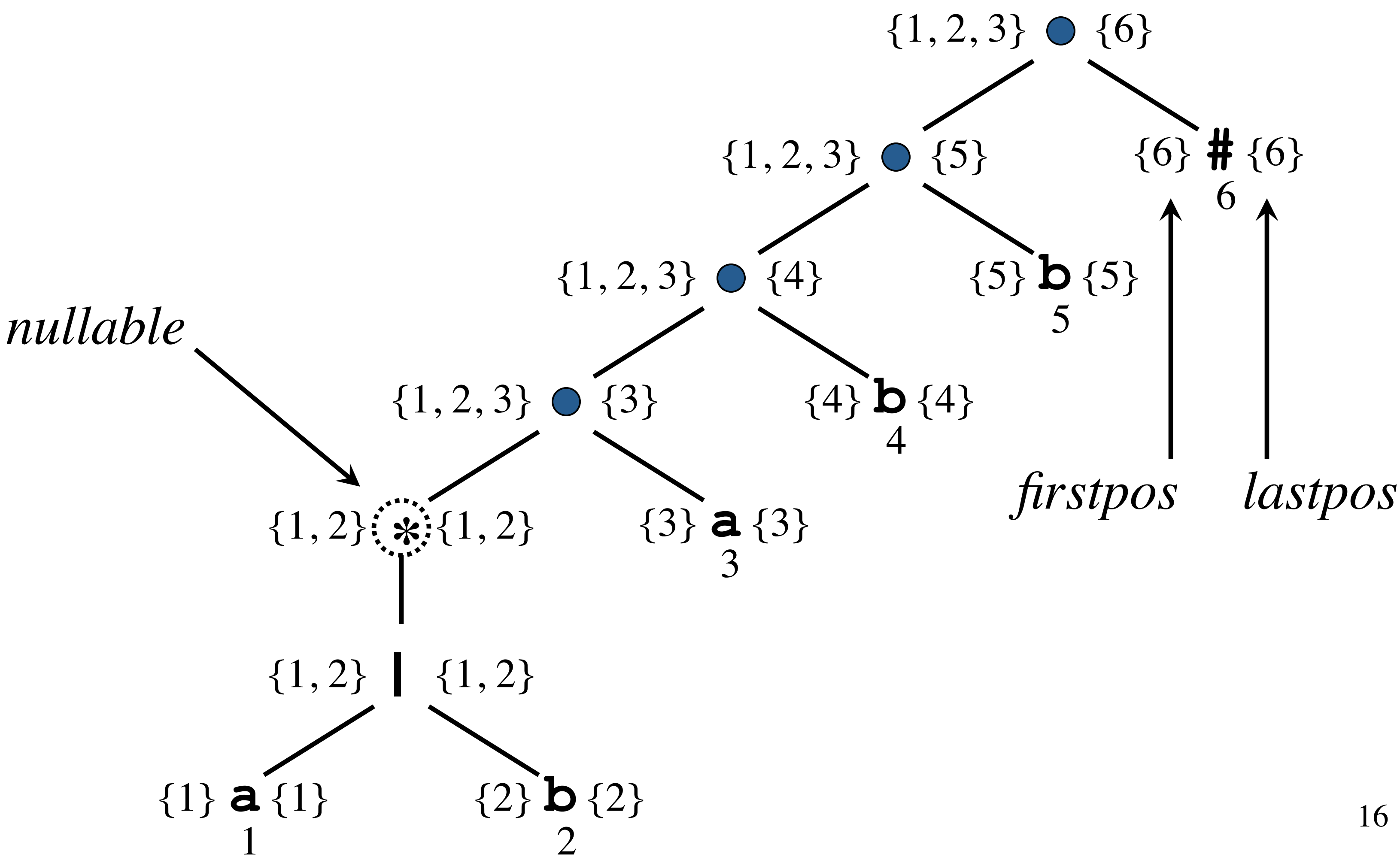
From Regular Expression to DFA Directly: Annotating the Tree

- Traverse the tree to construct functions *nullable*, *firstpos*, *lastpos*, and *followpos*
- For a node *n*, let *L(n)* be the language generated by the subtree with root *n*
- *nullable*(*n*): *L(n)* contains the empty string ε
- *firstpos*(*n*): set of *positions* under *n* that can match the first symbol of a string in *L(n)*
- *lastpos*(*n*): the set of *positions* under *n* that can match the last symbol of a string in *L(n)*
- *followpos*(*i*): the set of positions that can follow position *i* in any generated string

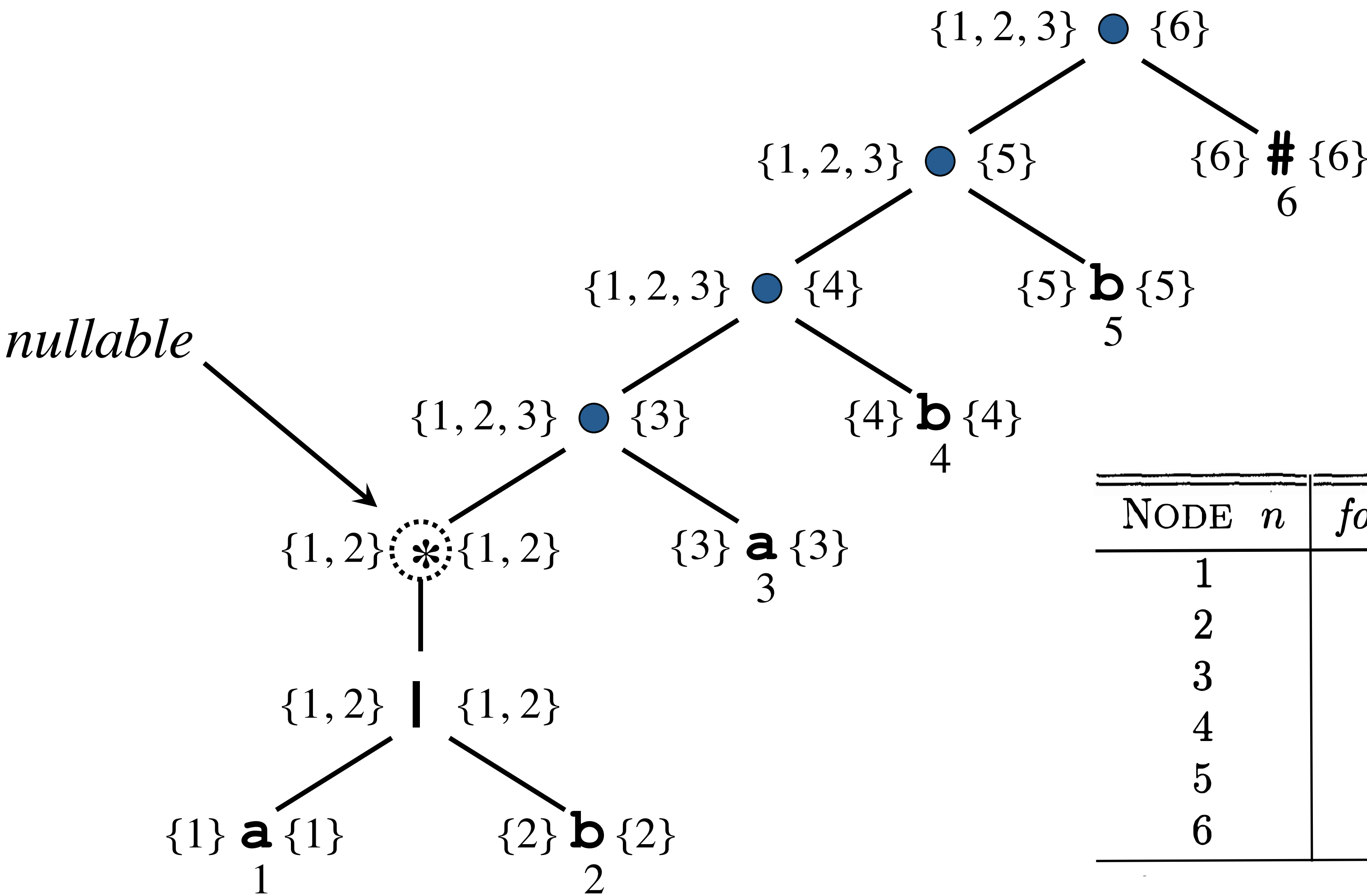
From Regular Expression to DFA Directly: Annotating the Tree

Node n	$nullable(n)$	$firstpos(n)$	$lastpos(n)$
Leaf ε	true	$\emptyset$	$\emptyset$
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	$firstpos(c_1)$	$lastpos(c_1)$

From Regular Expression to DFA
Annotating the Syntax Tree of (a|b)*abb#



From Regular Expression to DFA *followpos* on the Syntax Tree of (a|b)*abb#



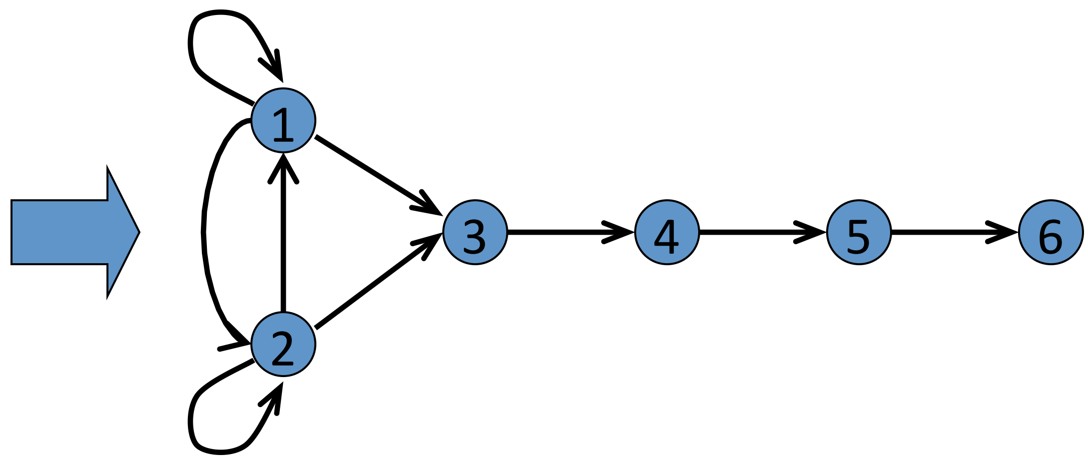
NODE n	*followpos*(n)
1	{1, 2, 3}
2	{1, 2, 3}
3	{4}
4	{5}
5	{6}
6	∅

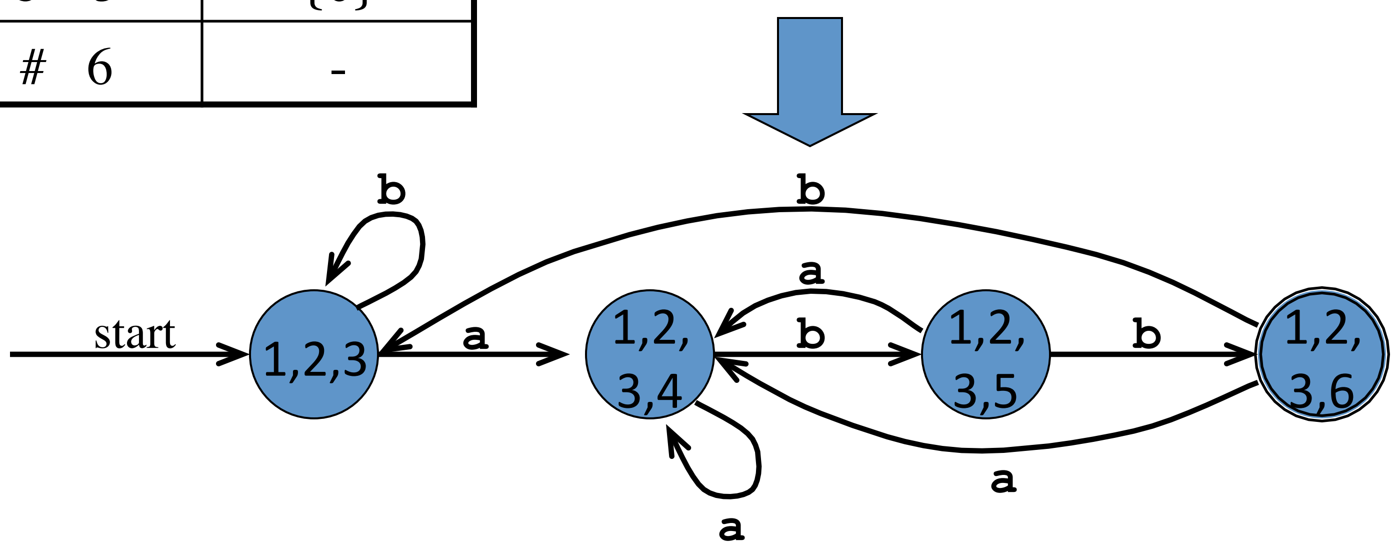
From Regular Expression to DFA Directly: *followpos*

```
for each node n in the tree do
    if n is a cat-node with left child c1 and right child c2 then
        for each i in lastpos(c1) do
            followpos(i) := followpos(i) ∪ firstpos(c2)
        end do
    else if n is a star-node
        for each i in lastpos(n) do
            followpos(i) := followpos(i) ∪ firstpos(n)
        end do
    end if
end do
```

From Regular Expression to DFA Directly: Example

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





From Regular Expression to DFA Directly: The Algorithm

s_0 := *firstpos*(*root*) where *root* is the root of the syntax tree for *(**r**)#*
Dstates := {s_0} and is unmarked
while there is an unmarked state *T* in *Dstates* **do**
    mark *T*
    for each input symbol $a \in \Sigma$ **do**
        let *U* be the union of *followpos*(*p*) for all positions *p* in *T*
            such that the symbol at position *p* is *a*
        if *U* is not empty and not in *Dstates* **then**
            add *U* as an unmarked state to *Dstates*
        end if
        Dtran[*T*, *a*] := *U*
    end do
end do