LEX5: Regexps to NFA

Lexical Analysis

CMPT 379: Compilers

Instructor: Anoop Sarkar

anoopsarkar.github.io/compilers-class

Building a Lexical Analyzer

- Token ⇒ Pattern
- Pattern ⇒ Regular Expression
- Regular Expression ⇒ NFA
- NFA ⇒ DFA
- DFA ⇒ Table-driven implementation of DFA

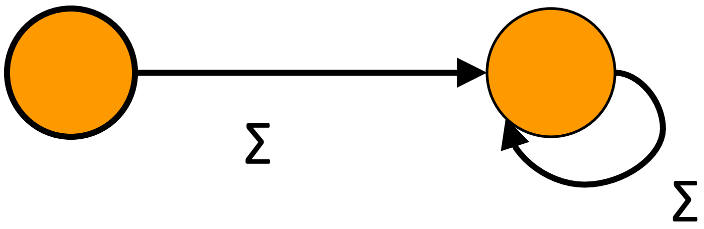
Thompson's construction

- Converts regexps to equivalent NFA
- Six simple rules
  - Empty language
  - Symbols (Σ)
  - Empty String (ε)
  - Alternation (r_1 or r_2)
  - Concatenation (r_1 followed by r_2)
  - Repetition (r_1*)

Used by Ken Thompson for pattern-based search in text editor QED (1968)

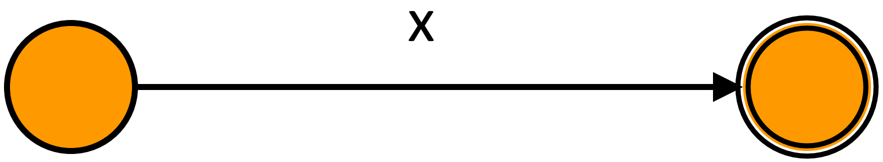
Thompson Rule 0

- For the empty language ϕ (optionally include a *sinkhole* state)



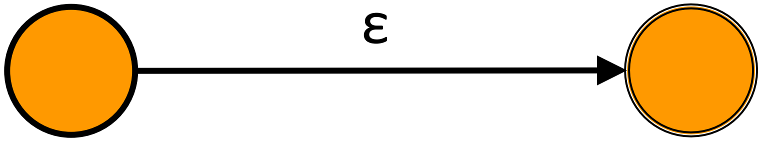
Thompson Rule 1

- For each symbol *x* of the alphabet, there is a NFA that accepts it



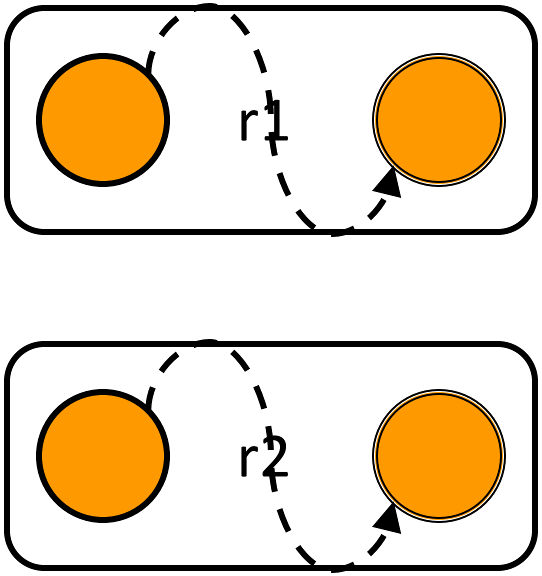
Thompson Rule 2

- There is an NFA that accepts only ε



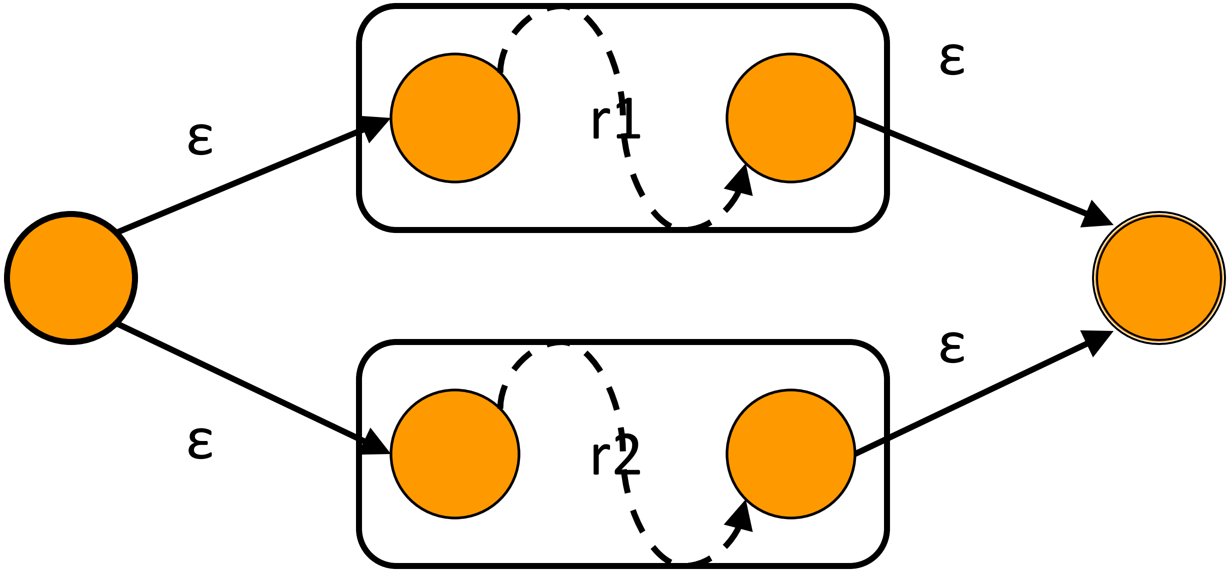
Thompson Rule 3

- Given two NFAs for r_1, r_2, there is a NFA that accepts $r_1|r_2$



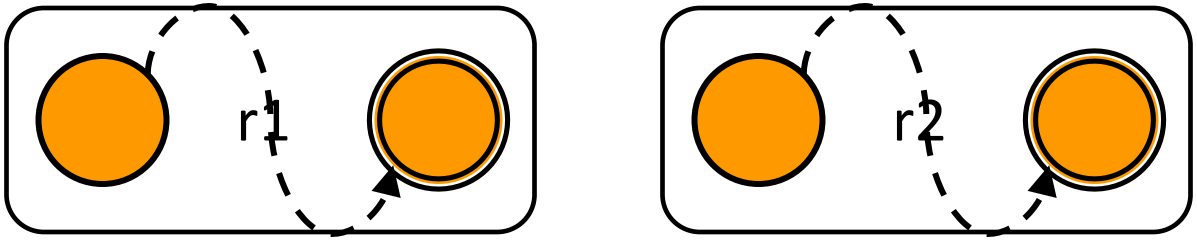
Thompson Rule 3

- Given two NFAs for r_1, r_2, there is a NFA that accepts $r_1|r_2$



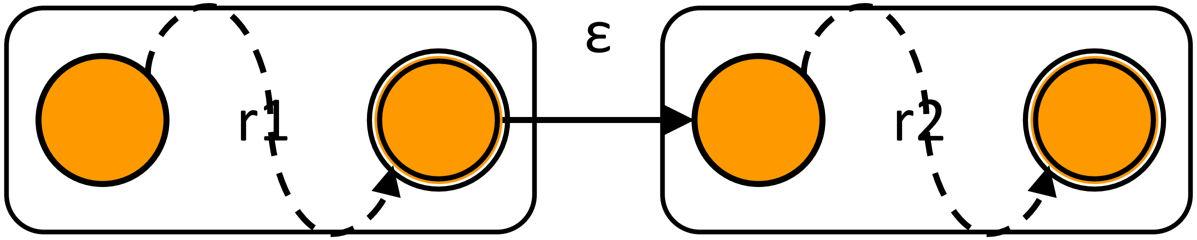
Thompson Rule 4

- Given two NFAs for r_1, r_2, there is a NFA that accepts r_1r_2



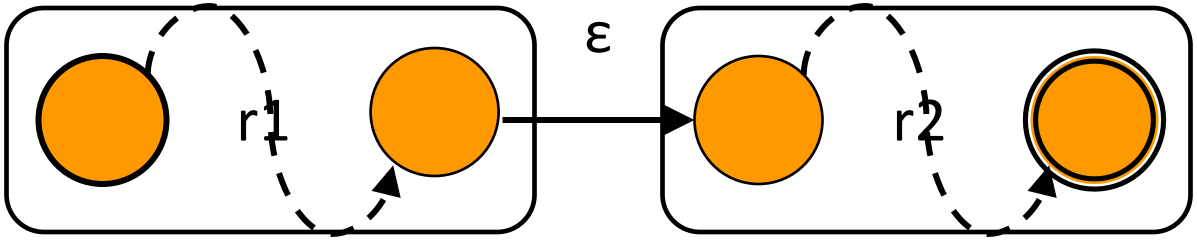
Thompson Rule 4

- Given two NFAs for r_1, r_2, there is a NFA that accepts r_1r_2



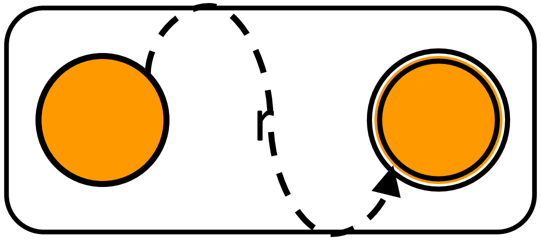
Thompson Rule 4

- Given two NFAs for r_1, r_2, there is a NFA that accepts r_1r_2



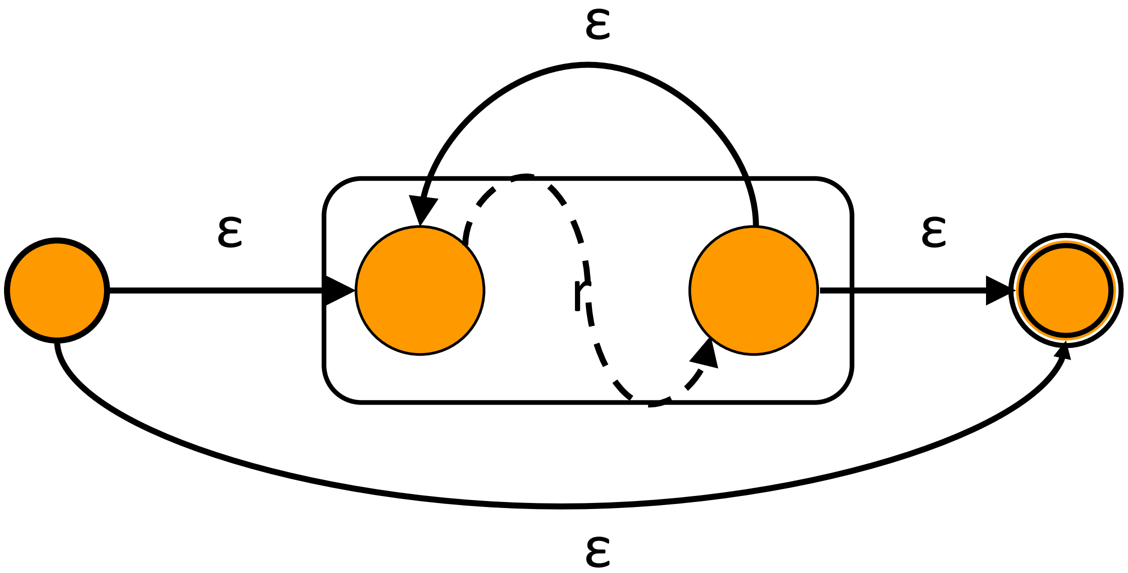
Thompson Rule 5

- Given a NFA for r, there is an NFA that accepts *r**



Thompson Rule 5

- Given a NFA for r, there is an NFA that accepts *r**



Example

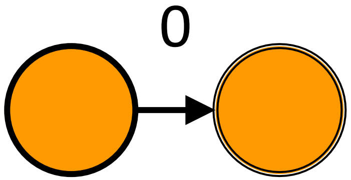
- Set of all binary strings that are divisible by four (include 0 in this set)
- Defined by the regexp: ((0|1)*00) | 0
- Apply Thompson's Rules to create an NFA

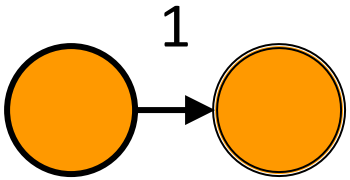
Basic Blocks 0 and 1

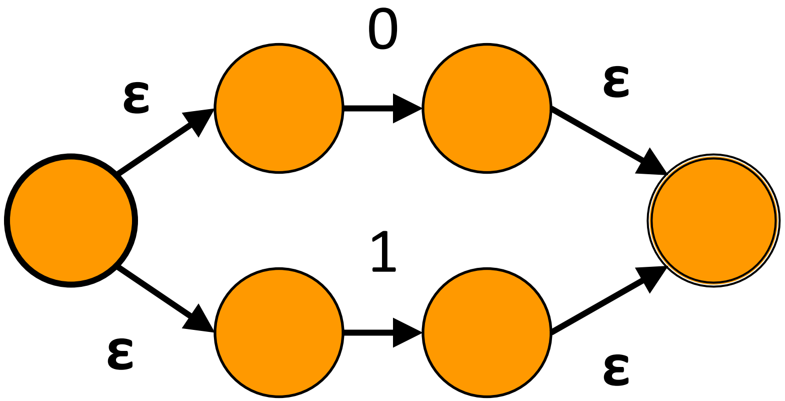
((0|1)*00) | 0

NFA for 0

NFA for 1

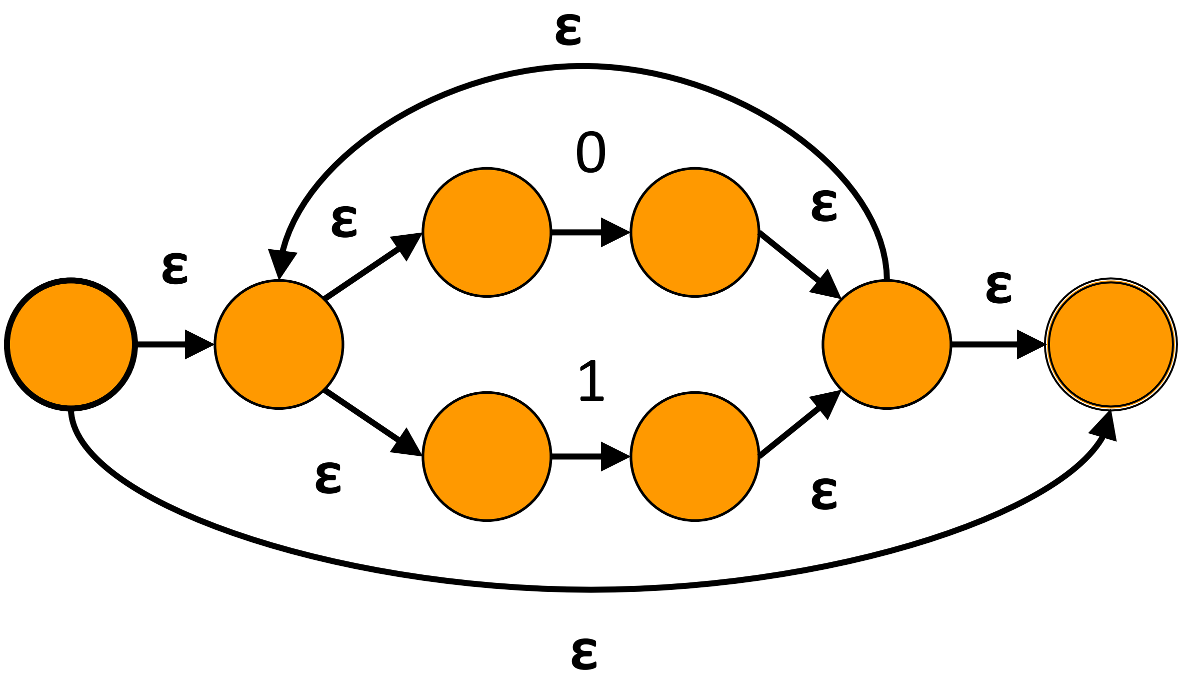






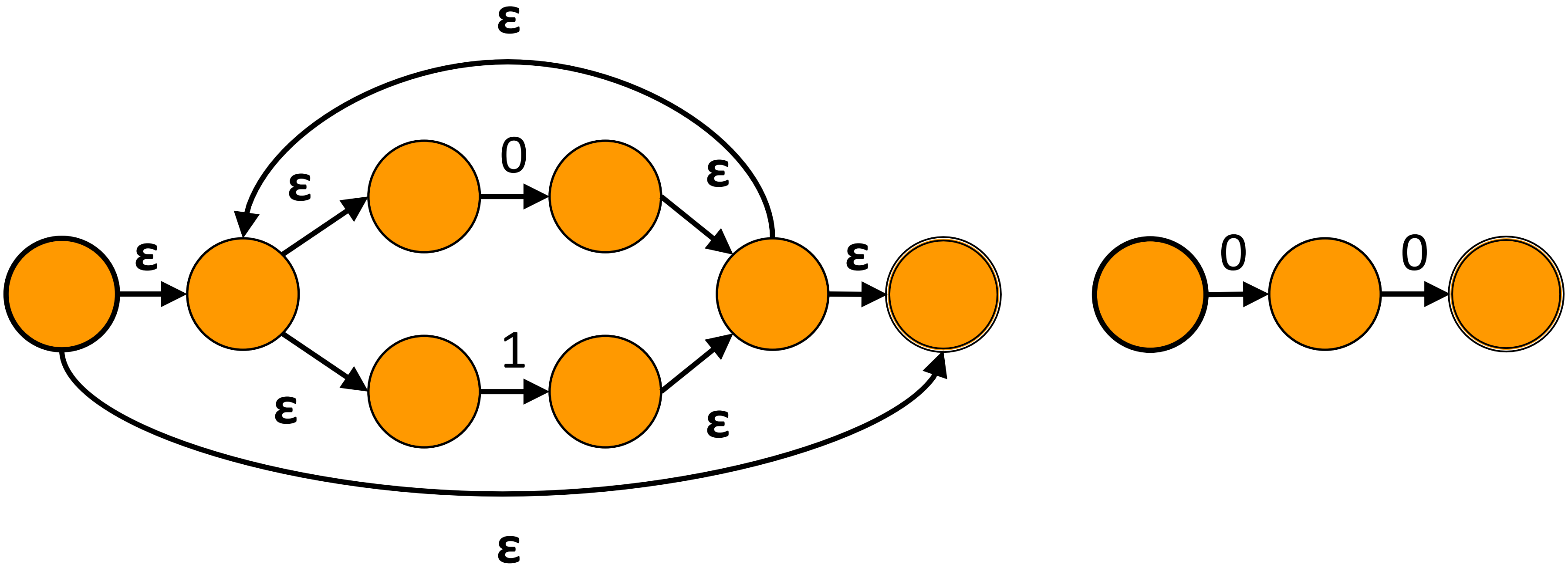
0|1

((0|1)*00) | 0



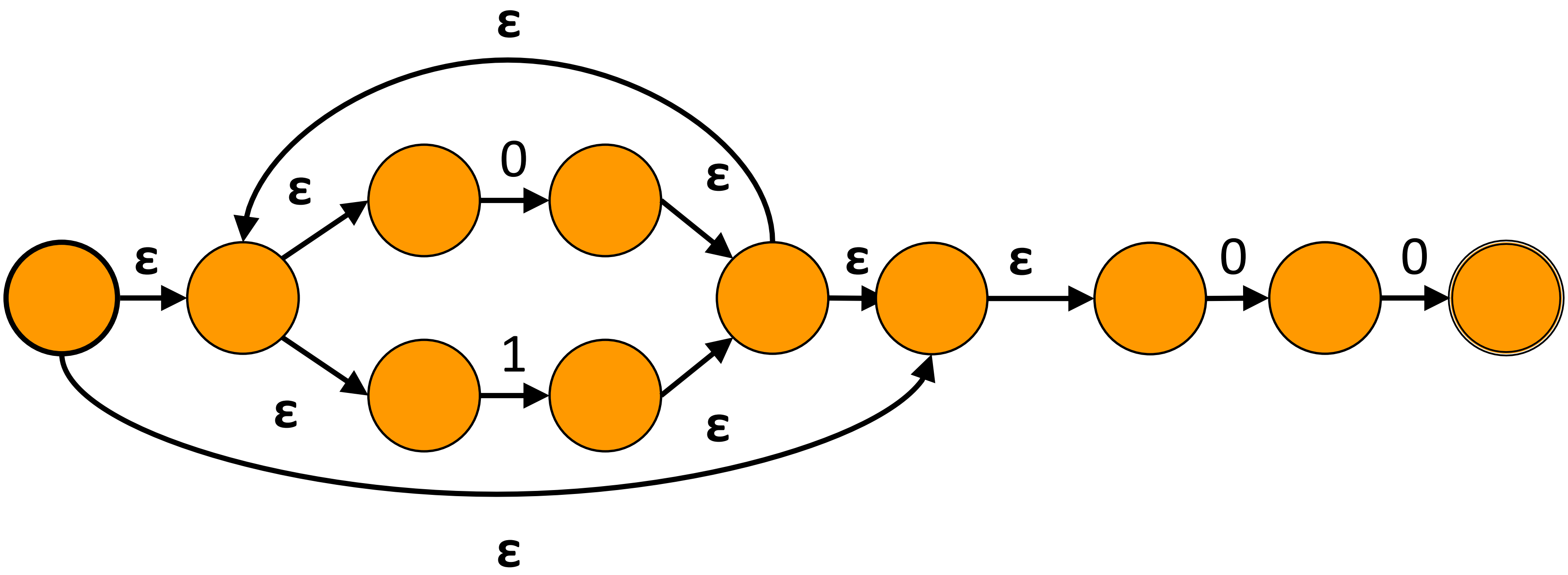
(0|1)*

((0|1)*00) | 0



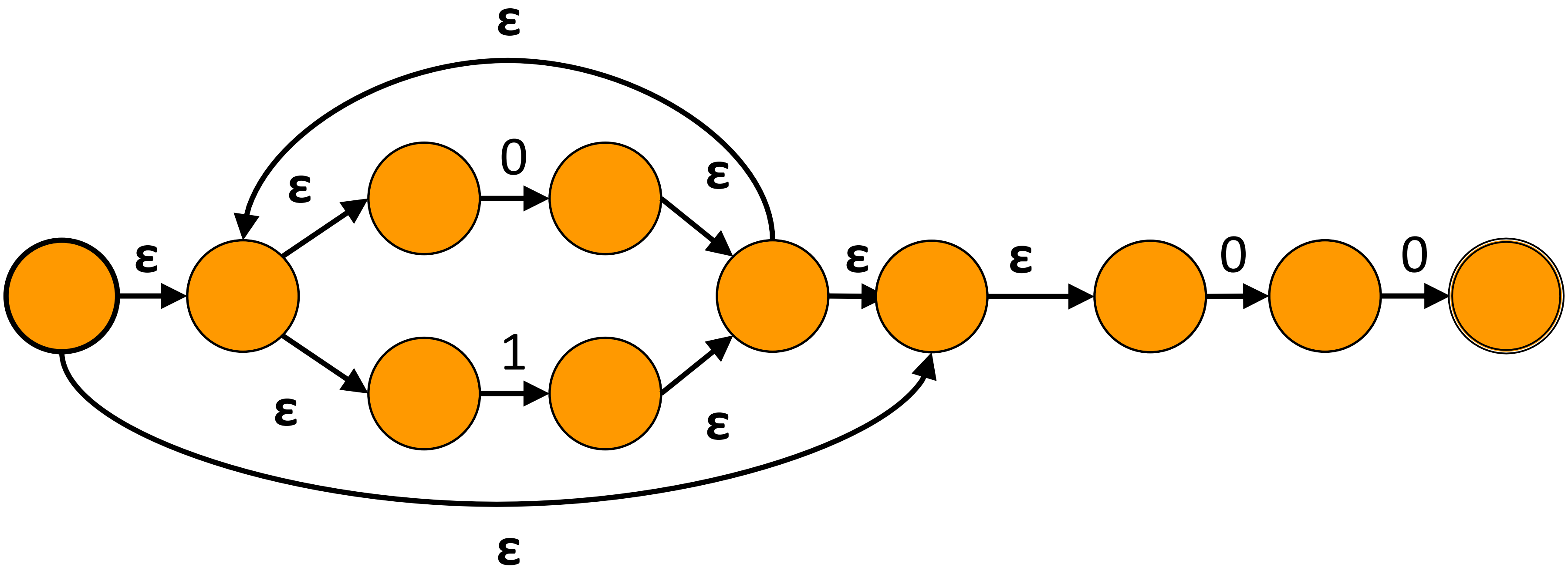
(0|1)*00

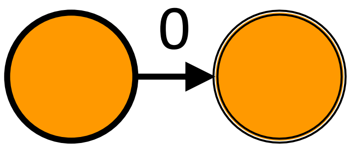
((0|1)*00) | 0



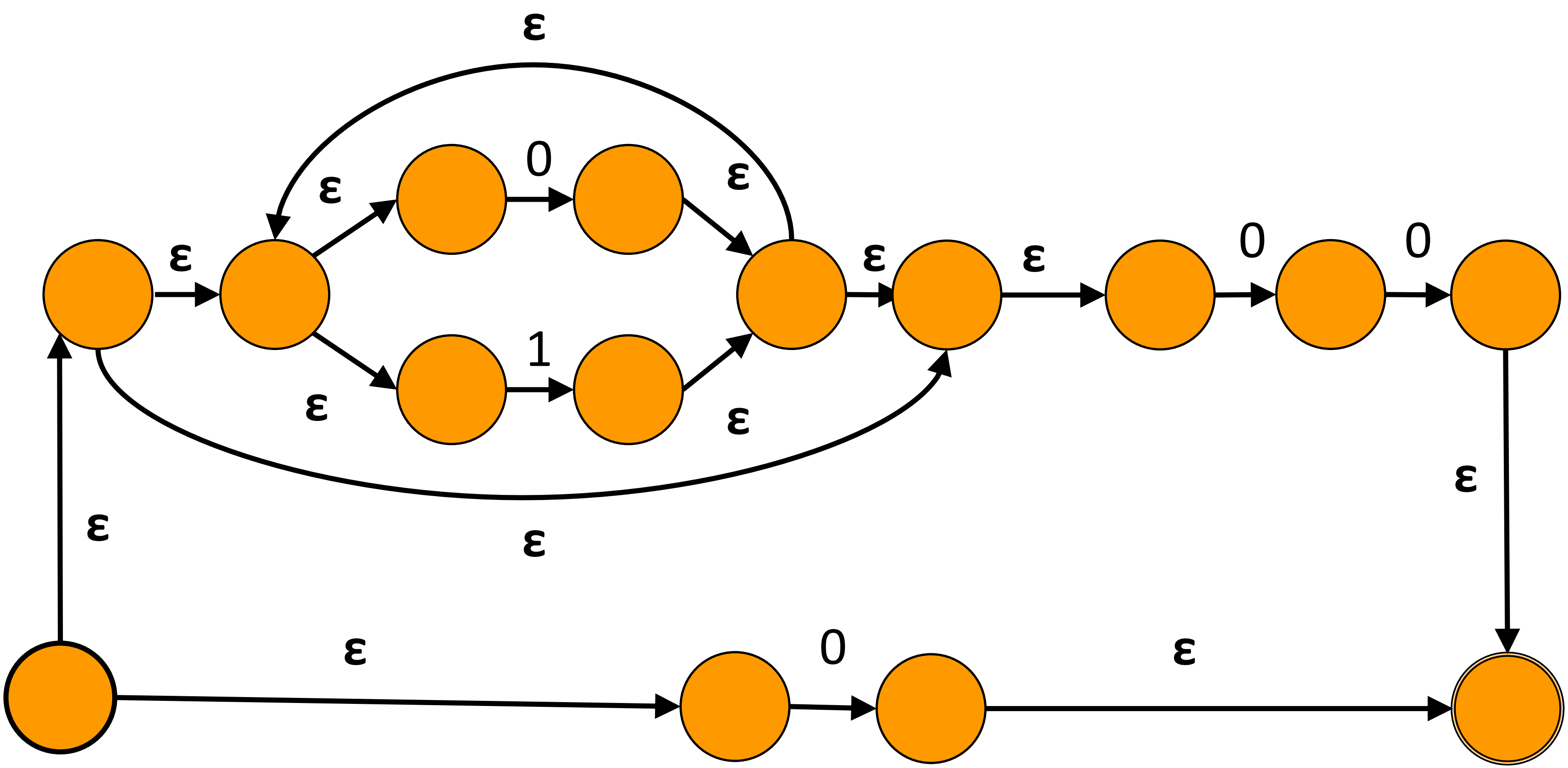
(0|1)*00

((0|1)*00) | 0





((0|1)*00)|0



((0|1)*00)|0

Thompson’s construction

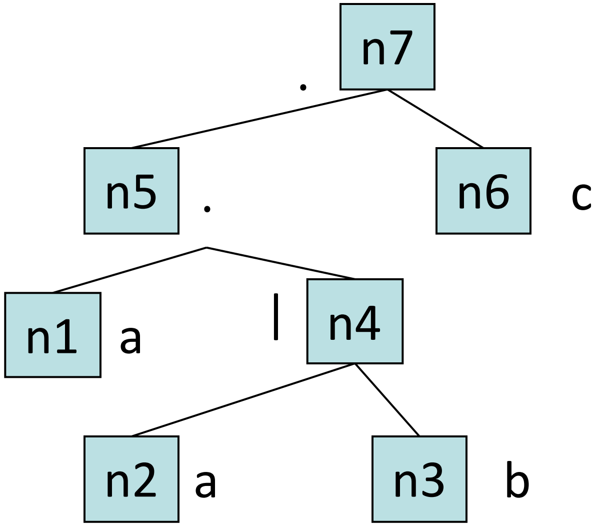
Converts regexps to NFA

Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

Post-order traversal of regexp tree



n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson's construction

Converts regexps to NFA

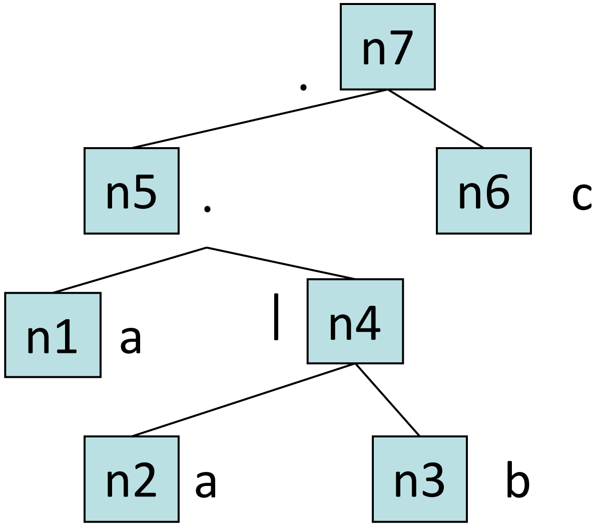
Build NFA recursively from regexp tree

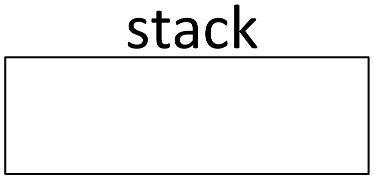
(a(a|b))c

aab|.c.

↑

Post-order traversal of regexp tree





n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

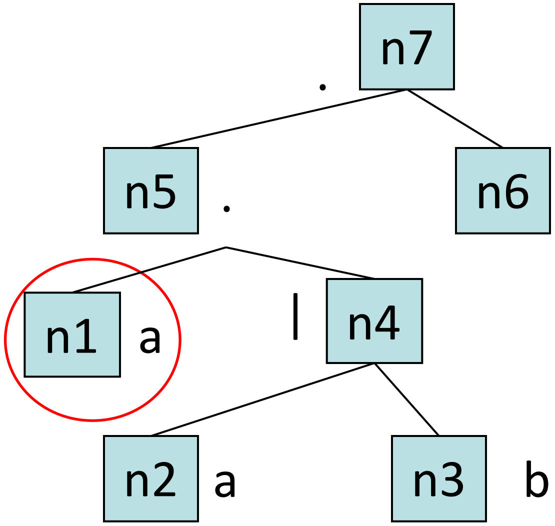
Converts regexps to NFA

Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

↑



push n1

stack

n1

Post-order traversal of regexp tree

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

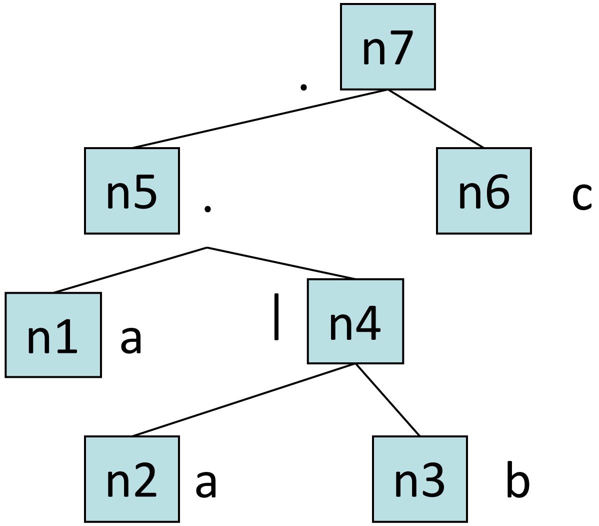
Converts regexps to NFA

Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

Post-order traversal of regexp tree



stack

n1

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

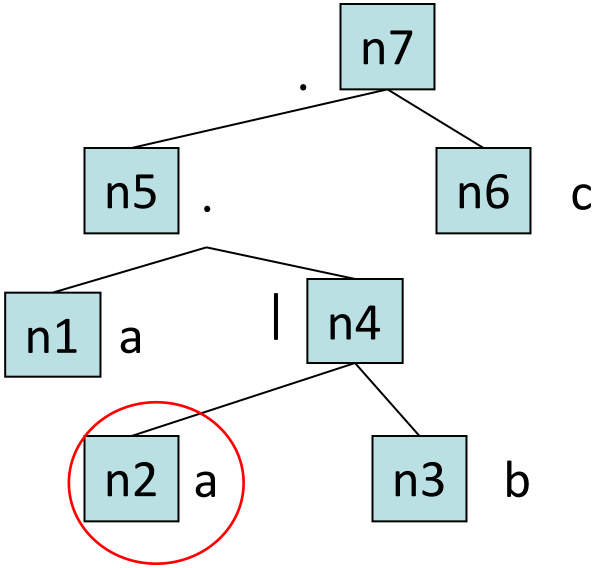
Converts regexps to NFA

Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

Post-order traversal of regexp tree



push n2

stack

n2, n1

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

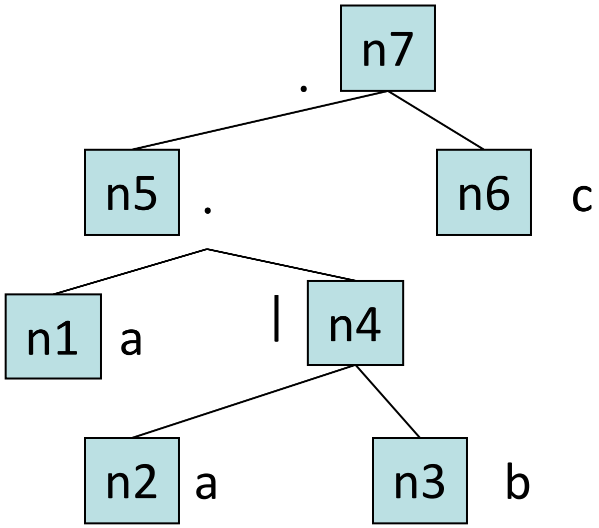
Converts regexps to NFA

Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

Post-order traversal of regexp tree



stack

n2, n1

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

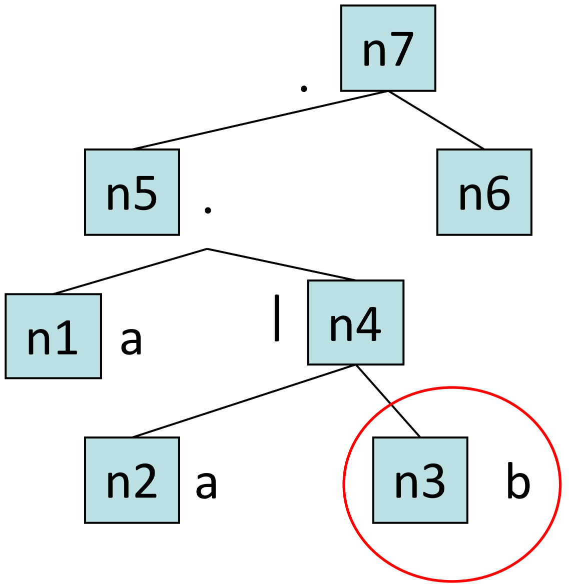
Converts regexps to NFA

Build NFA recursively from regexp tree

(a(a|b))c

Post-order traversal of regexp tree

aab|.c.



push n3

stack

n3, n2, n1

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

Converts regexps to NFA

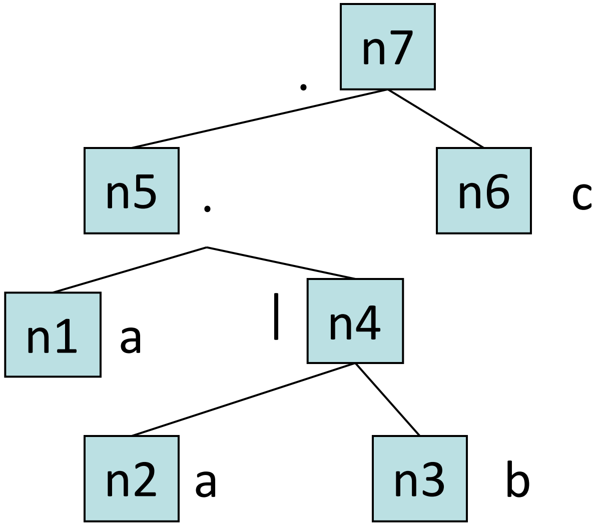
Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

↑

Post-order traversal of regexp tree



stack

n3, n2, n1

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

Converts regexps to NFA

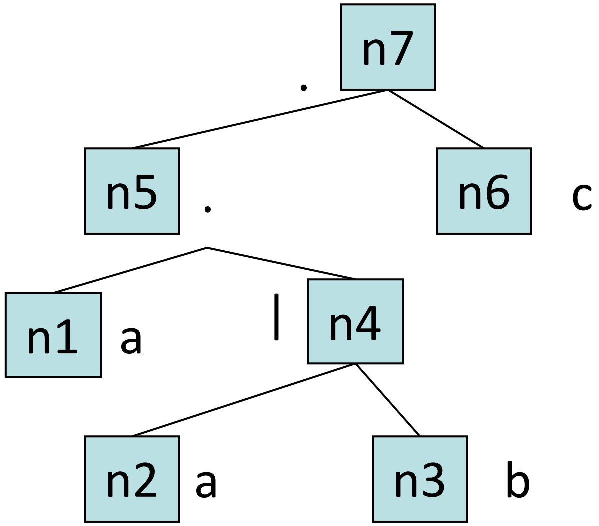
Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

↑

Post-order traversal of regexp tree



pop n3,n2

stack

n1

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

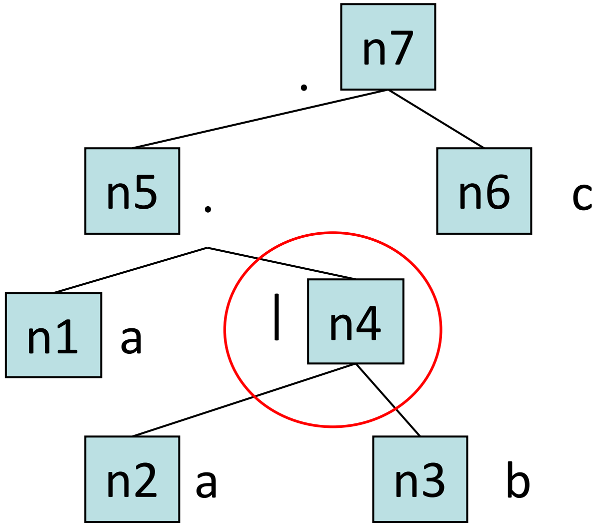
Converts regexps to NFA

Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

Post-order traversal of regexp tree



push n4

stack

n4, n1

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

Converts regexps to NFA

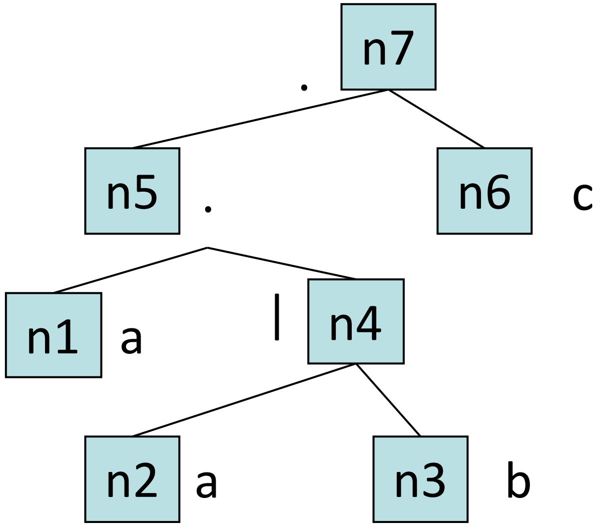
Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

↑

Post-order traversal of regexp tree



stack

n4, n1

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

Converts regexps to NFA

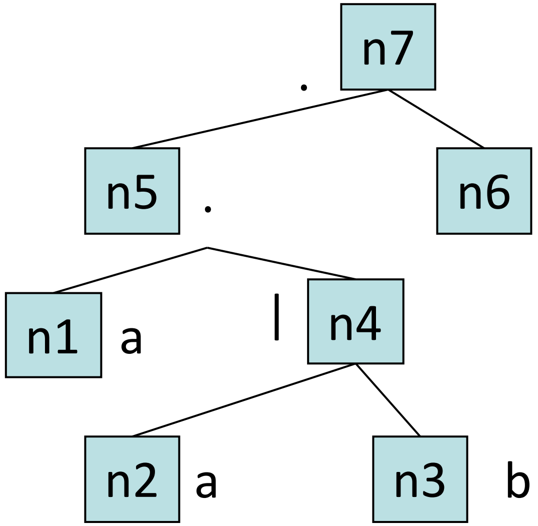
Build NFA recursively from regexp tree

(a(a|b))c

Post-order traversal of regexp tree

aab|.c.

↑



pop n4,n1

stack

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson's construction

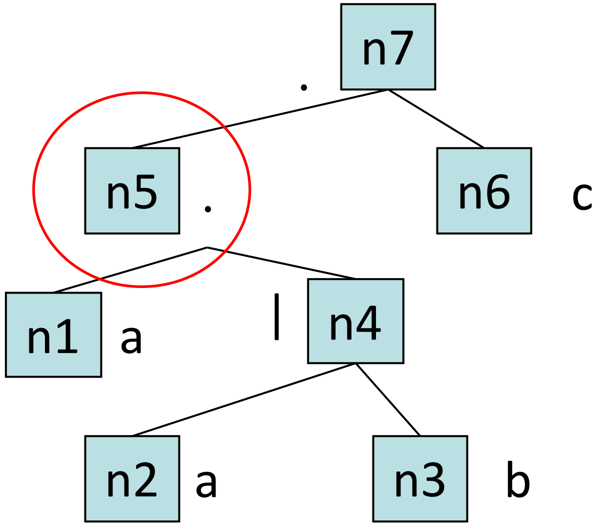
Converts regexps to NFA

Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

Post-order traversal of regexp tree



push n5

stack

n5

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

Converts regexps to NFA

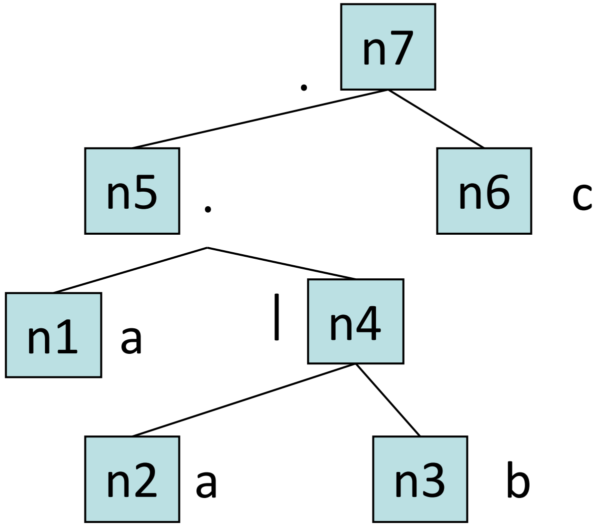
Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

↑

Post-order traversal of regexp tree



stack

n5

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson's construction

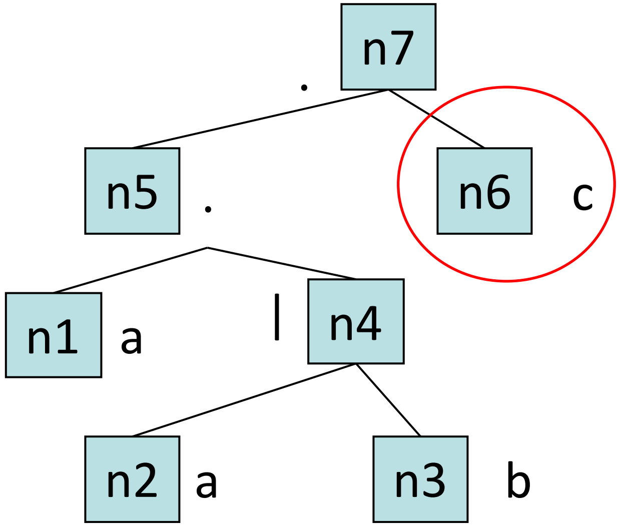
Converts regexps to NFA

Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

↑



push n6

stack

n6, n5

Post-order traversal of regexp tree

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

Converts regexps to NFA

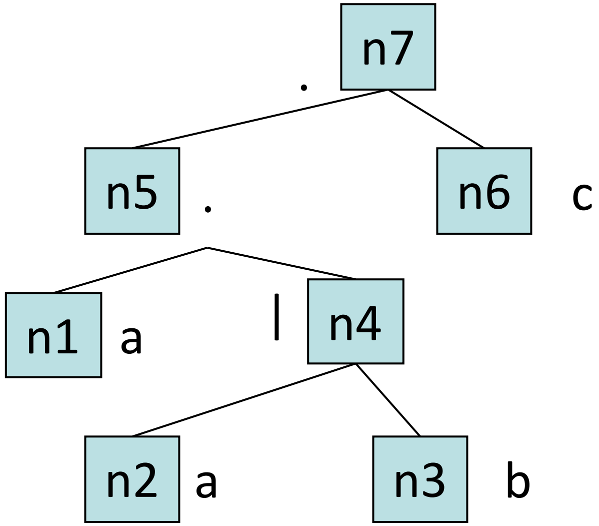
Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

↑

Post-order traversal of regexp tree



stack

n6, n5

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

Converts regexps to NFA

Build NFA recursively from regexp tree

(a(a|b))c

aab|.c.

Post-order traversal of regexp tree

n1= nfa(a)

n2= nfa(a)

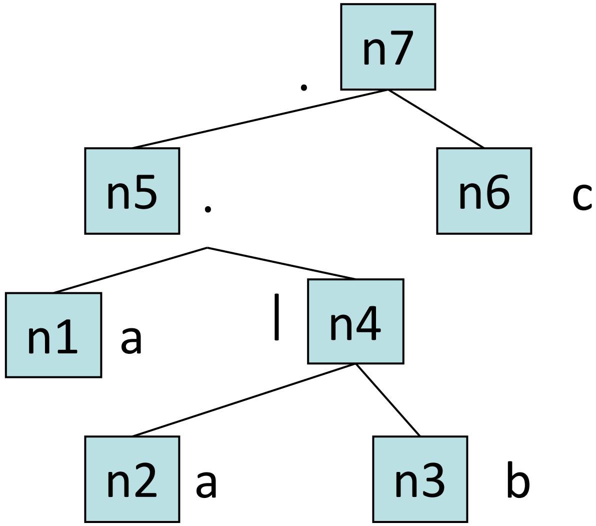
n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)



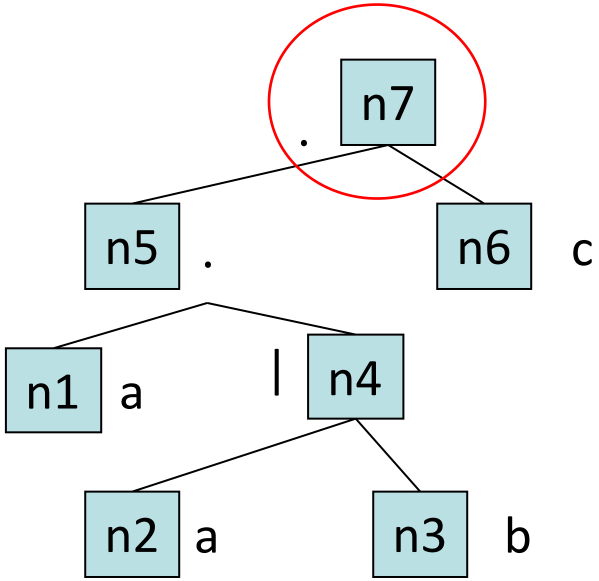
pop n6, n5

stack

Thompson’s construction

Converts regexps to NFA

Build NFA recursively from regexp tree



(a(a|b))c

aab|.c.

↑

push n7

stack

n7

Post-order traversal of regexp tree

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

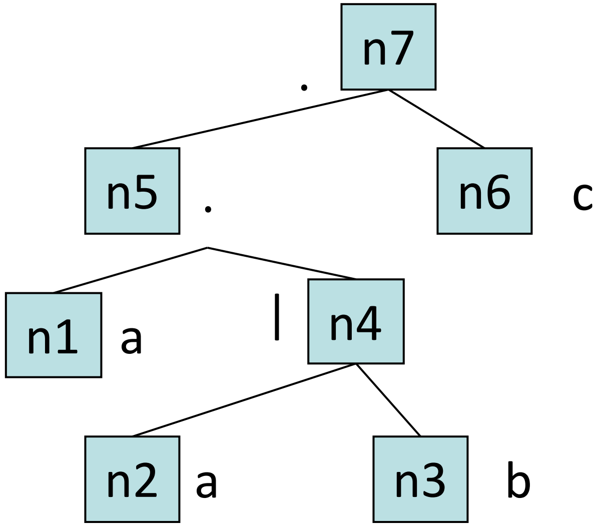
n6= nfa(c)

n7= nfa(n5, n6, .)

Thompson’s construction

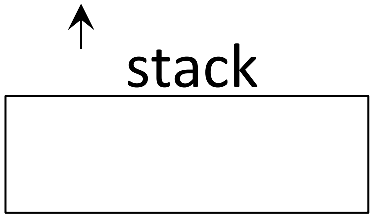
Converts regexps to NFA

Build NFA recursively from regexp tree



(a(a|b))c

aab|.c.



pop n7

Post-order traversal of regexp tree

n1= nfa(a)

n2= nfa(a)

n3= nfa(b)

n4= nfa(n2, n3, |)

n5= nfa(n1, n4, .)

n6= nfa(c)

n7= nfa(n5, n6, .)