Simulating a DFA

```
function Match () returns boolean
  var s: State
      ch: char
  s = s0
  ch = nextChar()
  while ch ≠ EOF do
    s = Move(s,ch)
    ch = NextChar()
  endWhile
  if s ∈ FinalStates then
    return true
  else
    return false
  endIf
endFunction
```

i.e., "int"

The "Move" function

Perhaps an array `s = Move[s,ch]`

Perhaps a linked list representation, to save space

Is Move always defined?

Use "dead" state to deal with undefined edges.

Simulating a NFA

States: s, t

Sets of states: S, T

Deterministic Machine:

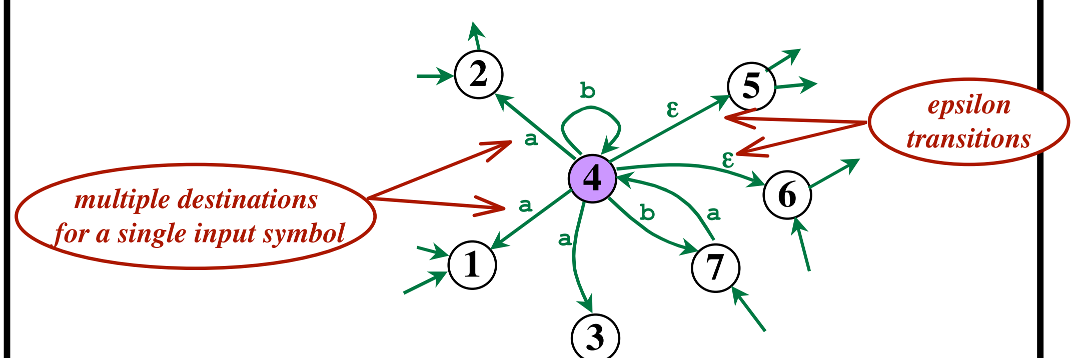
`Move(s,ch) → t`

Non-deterministic Machine:

`Move`$_{NFA}$`(S,ch) → T`

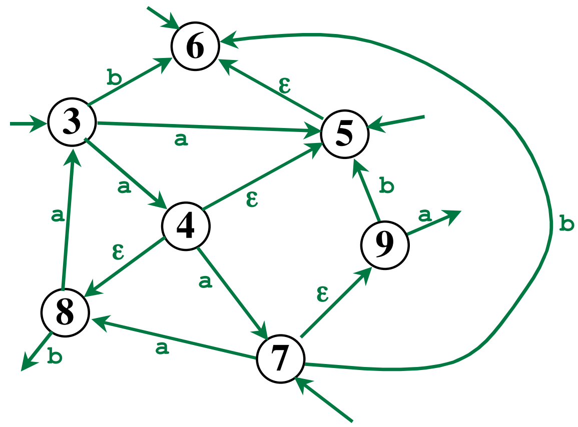
If s ∈ S and there is an edge... $s \xrightarrow{ch} t$

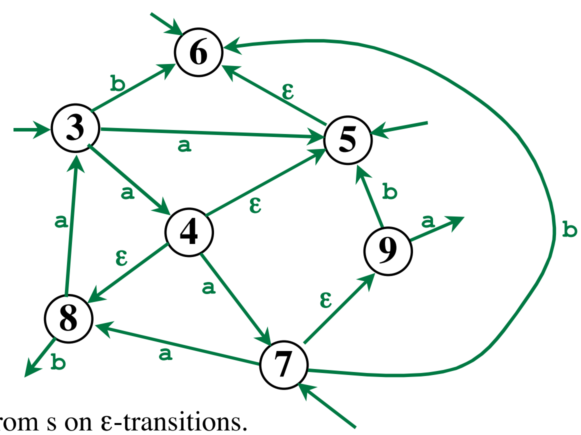
then t ∈ T



Example

$Move_{NFA}$ ({3,7}, a) = { 4, 5, 8 }





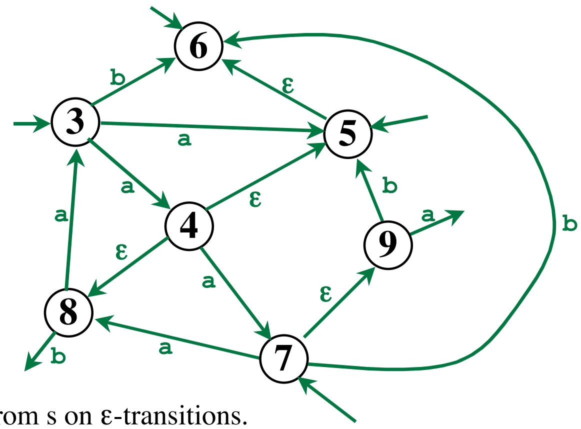
ε-Closure

Define ε-Closure (s):

The set of states reachable from s on ε-transitions.

ε-closure (4) = { 4, 5, 6, 8 }

ε-Closure



Define ε-Closure (s):

The set of states reachable from s on ε-transitions.

ε-closure (4) = { 4, 5, 6, 8 }

Define ε-Closure(S):

{ t | t ∈ ε-closure (s) for all s ∈ S }

ε-closure ({4, 7}) = { 4, 5, 6, 7, 8, 9 }

Computation of ε-Closure

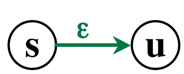
Given: T (= a set of states)

Goal: Compute ε-Closure(T)

The textbook presents a different algorithm!

Approach: Use a stack of states (= the states that we still need to look at)

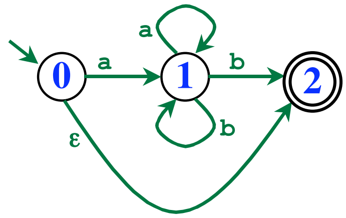
Algorithm:

```
var
  stack: stack of states
  result: set of states
push all states in T onto stack
result = T
while stack not empty do
  s = pop(stack)
  for each state u
            such that an edge
```



```
                                  exists do
    if u is not in result then
      add u to result
      push u onto stack
    endIf
  endFor
endWhile
```

Example

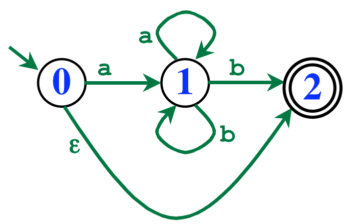
Input String: **abab**



Let S be the state(s) we are in...

Example

Input String: **abab**

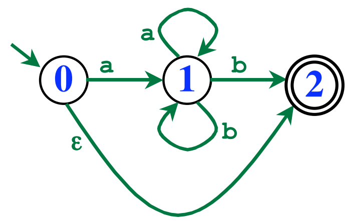


Let S be the state(s) we are in...

S = ε-Closure ({0})

Example

Input String: `abab`



Let S be the state(s) we are in...

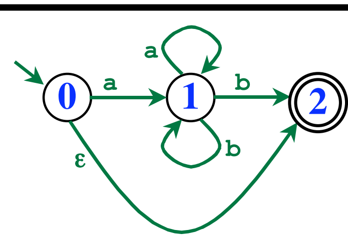
S = ε-Closure ({0})

= {0,2}

Lexical Analysis - Part 2

Example

Input String: `abab`



Let S be the state(s) we are in...

S = ε-Closure ({0})

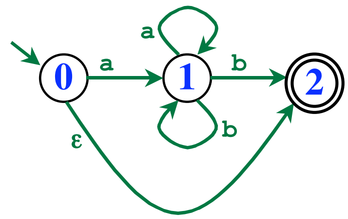
= {0,2}

Look at next character...

ch = `a`

Example

Input String: **abab**



Let S be the state(s) we are in...

S = ε-Closure ({0})

= {0,2}

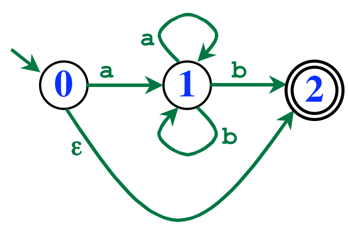
Look at next character...

ch = a

Move to next state(s)...

Example

Input String: **abab**



Let S be the state(s) we are in...

S = ε-Closure ({0})

= {0,2}

Look at next character...

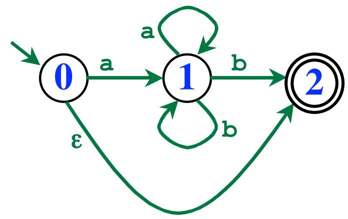
ch = a

Move to next state(s)...

S = ε-Closure ($Move_{NFA}$({0,2}, a)

Example

Input String: **abab**



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

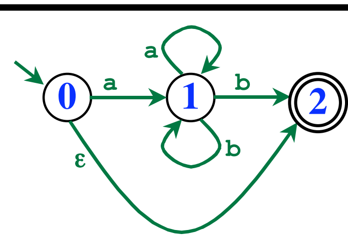
ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, a)
= ε-Closure ({1})

Example

Input String: **abab**



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

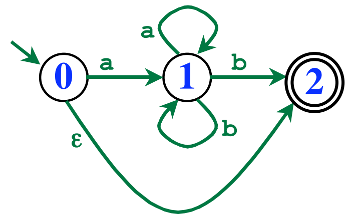
ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, a)
= ε-Closure ({1})
= {1}

Example

Input String: **abab**



Let S be the state(s) we are in...

S = ε-Closure ({0})

= {0,2}

Look at next character...

ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, a)

= ε-Closure ({1})

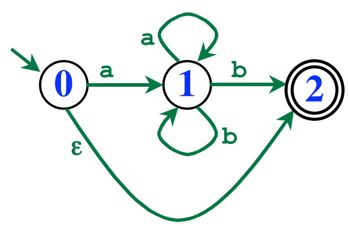
= {1}

Look at next character...

ch = b

Example

Input String: **abab**



Let S be the state(s) we are in...

S = ε-Closure ({0})

= {0,2}

Look at next character...

ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, a)

= ε-Closure ({1})

= {1}

Look at next character...

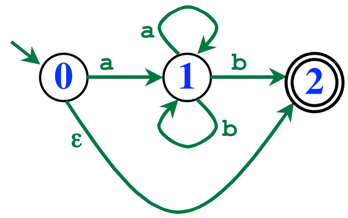
ch = b

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, b)

Example

Input String: `abab`



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

ch = `a`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, `a`)
= ε-Closure ({1})
= {1}

Look at next character...

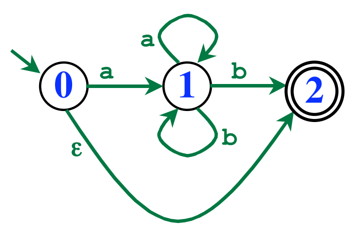
ch = `b`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, `b`)
= ε-Closure ({1,2})

Example

Input String: `abab`



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

ch = `a`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, `a`)
= ε-Closure ({1})
= {1}

Look at next character...

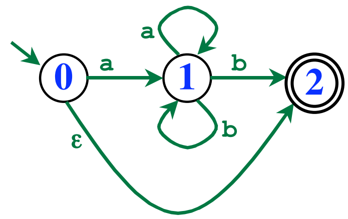
ch = `b`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, `b`)
= ε-Closure ({1,2})
= {1,2}

Example

Input String: **abab**



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, a)
= ε-Closure ({1})
= {1}

Look at next character...

ch = b

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, b)
= ε-Closure ({1,2})
= {1,2}

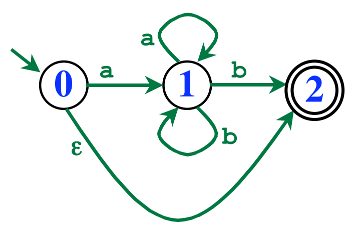
Look at next character...

ch = a

Lexical Analysis - Part 2

Example

Input String: **abab**



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, a)
= ε-Closure ({1})
= {1}

Look at next character...

ch = b

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, b)
= ε-Closure ({1,2})
= {1,2}

Look at next character...

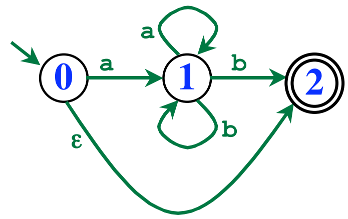
ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1,2}, a)

Example

Input String: `abab`



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

ch = `a`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, `a`)
= ε-Closure ({1})
= {1}

Look at next character...

ch = `b`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, `b`)
= ε-Closure ({1,2})
= {1,2}

Look at next character...

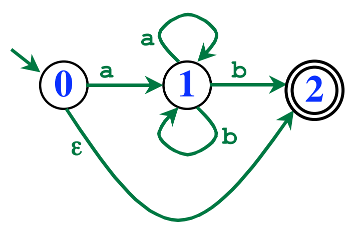
ch = `a`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1,2}, `a`)
= ε-Closure ({1})

Example

Input String: `abab`



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

ch = `a`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, `a`)
= ε-Closure ({1})
= {1}

Look at next character...

ch = `b`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, `b`)
= ε-Closure ({1,2})
= {1,2}

Look at next character...

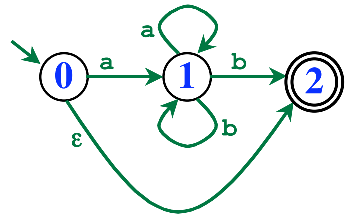
ch = `a`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1,2}, `a`)
= ε-Closure ({1})
= {1}

Example

Input String: `abab`



Let S be the state(s) we are in...

$S = \varepsilon\text{-Closure}(\{0\})$
$= \{0,2\}$

Look at next character...

ch = `a`

Move to next state(s)...

$S = \varepsilon\text{-Closure}(Move_{NFA}(\{0,2\}, \texttt{a}))$
$= \varepsilon\text{-Closure}(\{1\})$
$= \{1\}$

Look at next character...

ch = `b`

Move to next state(s)...

$S = \varepsilon\text{-Closure}(Move_{NFA}(\{1\}, \texttt{b}))$
$= \varepsilon\text{-Closure}(\{1,2\})$
$= \{1,2\}$

Look at next character...

ch = `a`

Move to next state(s)...

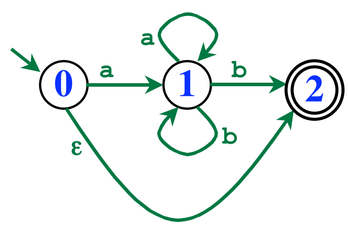
$S = \varepsilon\text{-Closure}(Move_{NFA}(\{1,2\}, \texttt{a}))$
$= \varepsilon\text{-Closure}(\{1\})$
$= \{1\}$

Look at next character...

ch = `b`

Example

Input String: `abab`



Let S be the state(s) we are in...

$S = \varepsilon\text{-Closure}(\{0\})$
$= \{0,2\}$

Look at next character...

ch = `a`

Move to next state(s)...

$S = \varepsilon\text{-Closure}(Move_{NFA}(\{0,2\}, \texttt{a}))$
$= \varepsilon\text{-Closure}(\{1\})$
$= \{1\}$

Look at next character...

ch = `b`

Move to next state(s)...

$S = \varepsilon\text{-Closure}(Move_{NFA}(\{1\}, \texttt{b}))$
$= \varepsilon\text{-Closure}(\{1,2\})$
$= \{1,2\}$

Look at next character...

ch = `a`

Move to next state(s)...

$S = \varepsilon\text{-Closure}(Move_{NFA}(\{1,2\}, \texttt{a}))$
$= \varepsilon\text{-Closure}(\{1\})$
$= \{1\}$

Look at next character...

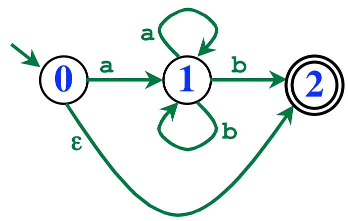
ch = `b`

Move to next state(s)...

$S = \varepsilon\text{-Closure}(Move_{NFA}(\{1\}, \texttt{b}) = \{1,2\}$

Example

Input String: **abab**



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, a)
= ε-Closure ({1})
= {1}

Look at next character...

ch = b

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, b)
= ε-Closure ({1,2})
= {1,2}

Look at next character...

ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1,2}, a)
= ε-Closure ({1})
= {1}

Look at next character...

ch = b

Move to next state(s)...

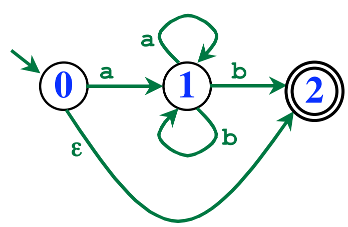
S = ε-Closure (Move$_{NFA}$({1}, b) = {1,2}

Look at next character...

ch = EOF

Example

Input String: **abab**



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, a)
= ε-Closure ({1})
= {1}

Look at next character...

ch = b

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, b)
= ε-Closure ({1,2})
= {1,2}

Look at next character...

ch = a

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1,2}, a)
= ε-Closure ({1})
= {1}

Look at next character...

ch = b

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, b) = {1,2}

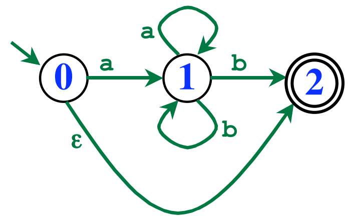
Look at next character...

ch = EOF

Does S contain a Final State?

Example

Input String: `abab`



Let S be the state(s) we are in...

S = ε-Closure ({0})
= {0,2}

Look at next character...

ch = `a`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({0,2}, `a`)
= ε-Closure ({1})
= {1}

Look at next character...

ch = `b`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, `b`)
= ε-Closure ({1,2})
= {1,2}

Look at next character...

ch = `a`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1,2}, `a`)
= ε-Closure ({1})
= {1}

Look at next character...

ch = `b`

Move to next state(s)...

S = ε-Closure (Move$_{NFA}$({1}, `b`) = {1,2}

Look at next character...

ch = `EOF`

Does S contain a Final State?

This string is accepted!!!

Simulating a NFA

```
function Match () returns boolean
  var S: set of states
      ch: char
  S = ε-Closure({s0})
  ch = nextChar()
  while ch ≠ EOF do
    S = ε-Closure(MoveNFA(S,ch))
    ch = NextChar()
  endWhile
  if S ∩ FinalStates ≠ {} then
    return true
  else
    return false
  endIf
endFunction
```

Thompson's Construction

Build an NFA for: `ab*c|d*e*`

Thompson's Construction

Build an NFA for: `ab*c|d*e*`

Break the expression into sub-expressions

$\underbrace{\texttt{(ab*c)}}_{\text{Build NFA for this}} \;\texttt{|}\; \underbrace{\texttt{(d*e*)}}_{\text{Build NFA for this}}$

Thompson's Construction

Build an NFA for: `ab*c|d*e*`

Break the expression into sub-expressions

`(ab*c) | (d*e* )`

Build NFA for this | **Build NFA for this**

Glue the two NFAs together

Thompson's Construction

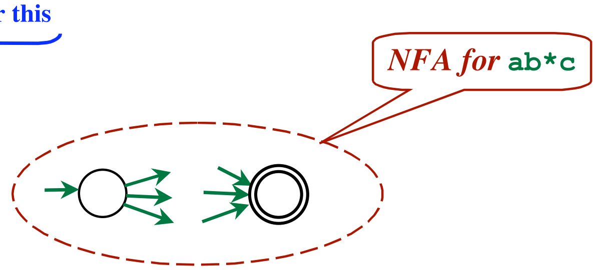
Build an NFA for: `ab*c|d*e*`

Break the expression into sub-expressions

`(ab*c) | (d*e* )`

Build NFA for this | **Build NFA for this**

Glue the two NFAs together



Thompson's Construction

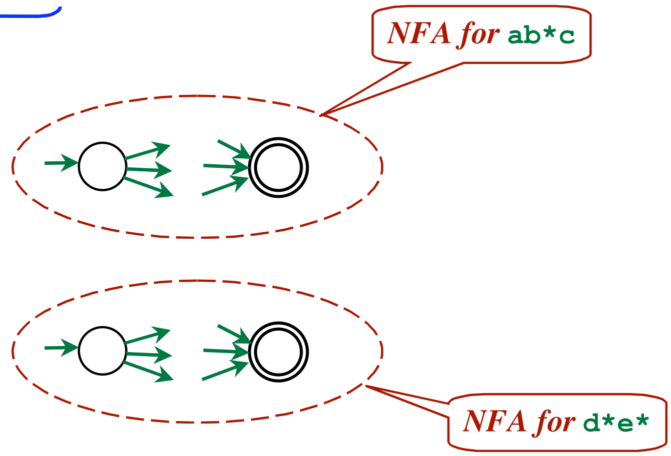
Build an NFA for: `ab*c|d*e*`

Break the expression into sub-expressions

`(ab*c) | (d*e* )`

Build NFA for this — Build NFA for this

Glue the two NFAs together



Thompson's Construction

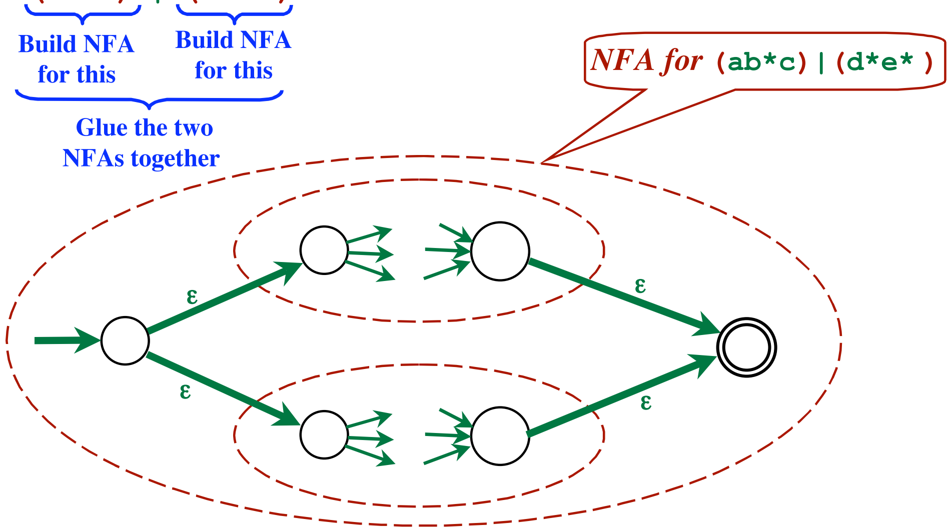
Build an NFA for: `ab*c|d*e*`

Break the expression into sub-expressions

`(ab*c) | (d*e* )`

Build NFA for this — Build NFA for this

Glue the two NFAs together



Thompson's Construction

Given:

Regular Expression, R

Goal:

Construct an NFA to recognize L(R)

Call the NFA which is constructed N(R)

Approach:

Look at the syntax of the expression R.

Top-most operator with sub-expressions:

$$R = R_1 \oplus R_2$$

For each sub-expression R_i...

Build an NFA called $N(R_i)$

For each larger expression

(...which is built from smaller expressions)

Build an NFA

using the NFA's for is component sub-expressions.

In other words, construct N(R) from $N(R_1)$ and $N(R_2)$

What kinds of regular expressions are there?

case 1: **a** where **a** $\in \Sigma$

case 2: $r_1 | r_2$

case 3: $r_1 r_2$

case 4: $r_1{}^*$

case 5: ε

case 6: (r_1)

Note:

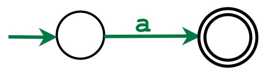
For every NFA we construct...

- 1 start state
- 1 accepting state
- No edge enters the start state
- No edge leaves the accepting state

Case 1: $\mathtt{a}$ where $\mathtt{a} \in \Sigma$

For a regular expression consisting of only $\mathtt{a}$ (for any $\mathtt{a} \in \Sigma$)

Construct

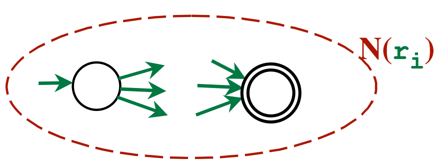


...and call it N($\mathtt{a}$)

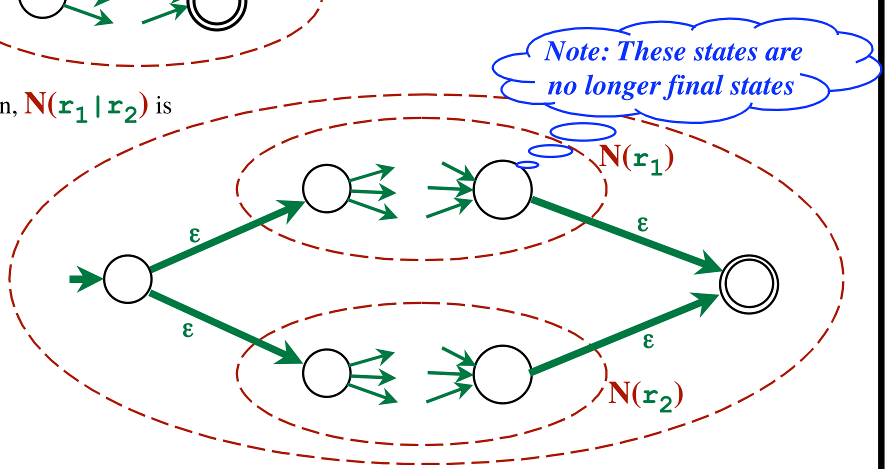
Case 2: $\mathtt{r_1 | r_2}$

For $\mathtt{r_1 | r_2}$, construct N($\mathtt{r_1 | r_2}$)

Let **N($\mathtt{r_i}$)** be:

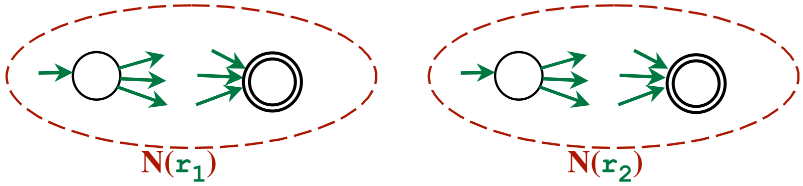


Then, **N($\mathtt{r_1 | r_2}$)** is

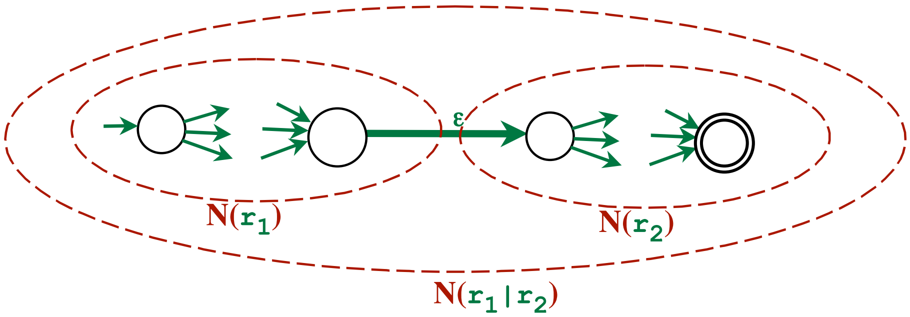


Case 3: $r_1 r_2$

From N(r_1) and N(r_2)...

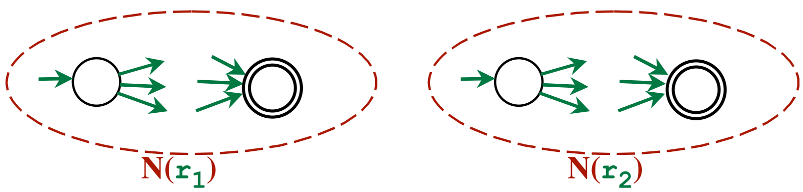


Construct **N($r_1 r_2$)** as follows:

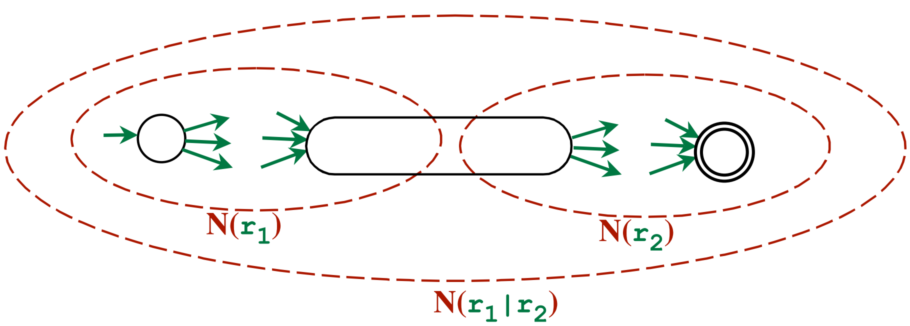


Case 3: $r_1 r_2$ *(alternative: combine states)*

From N(r_1) and N(r_2)...

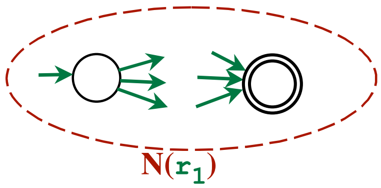


Construct **N($r_1 r_2$)** as follows:

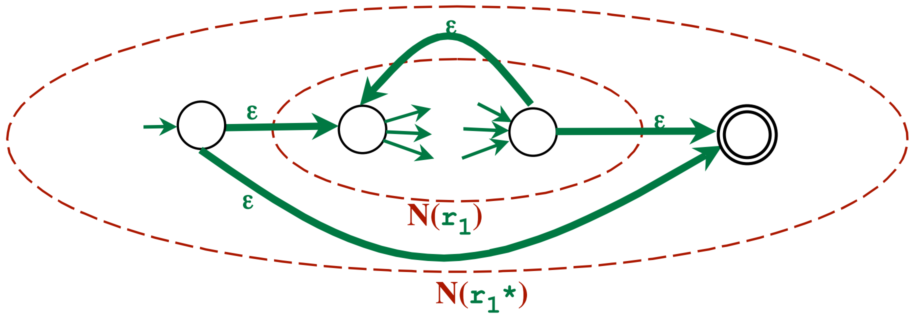


Case 4: r_1*

From N(r_1)...

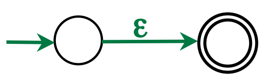


Construct N(r_1*) as follows:



Case 5: ε

Let N(ε) be...

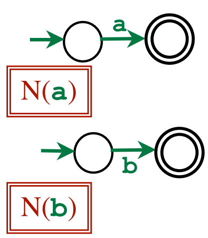


Case 6: (r_1)

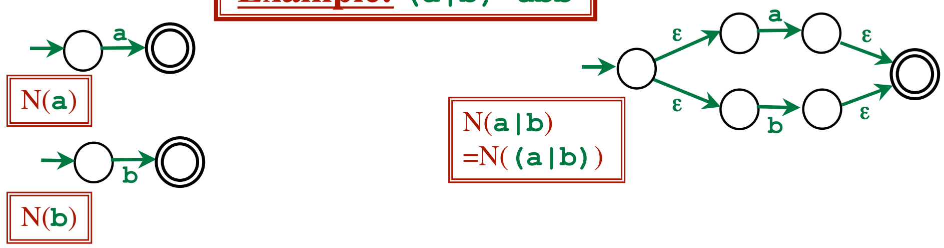
Let N((r_1)) be N(r_1) itself.

Example: (a|b)*abb

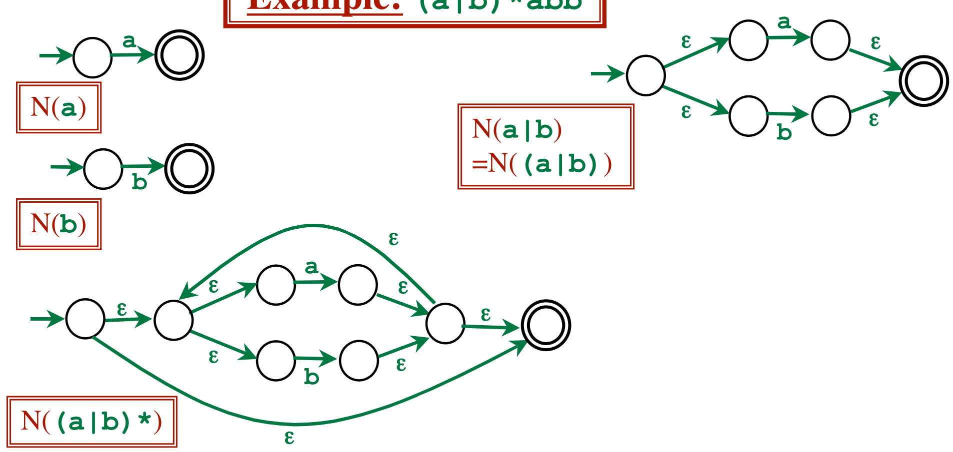
Example: (a|b)*abb



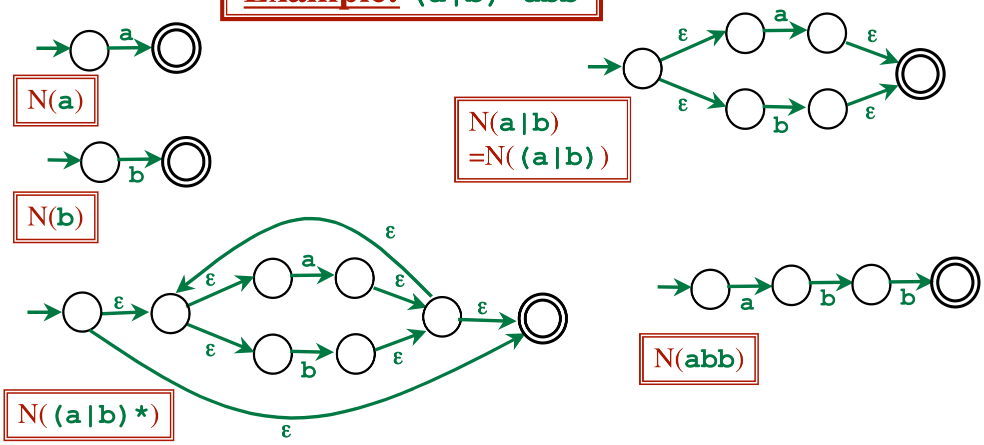
Example: (a|b)*abb



Example: (a|b)*abb



Example: (a|b)*abb



Example: (a|b)*abb