

Check-In #2

Review: Compiler Overview

What is an example of an input to a C compiler that would cause a lexical analysis error?

What is an example of an input to a C compiler that would cause a syntactic analysis error?

Check-In #2

Review: Compiler Overview

What is an example of an input to a C compiler that would cause a lexical analysis error?

What is an example of an input to a C compiler that would cause a syntactic analysis error?

Administriva

Housekeeping

Project 1

- Out tonight
- Add Flex rules for our language
- Might want to find a project partner!

Administriva

Housekeeping

What should we call our language?

Survey Results

Housekeeping

- Flipped Wednesdays?

EECS 665
COMPILER
CONSTRUCTION

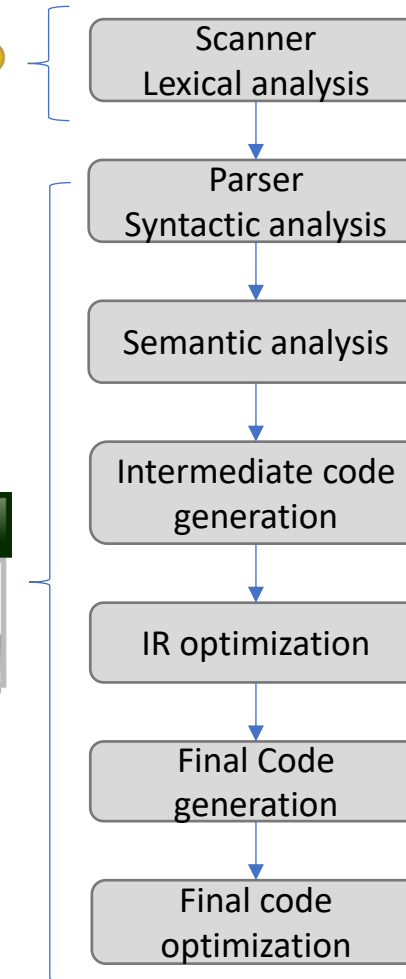
2 – Implementing Scanners

Compiler Construction

Progress Pics

Currently working on lexical analysis concepts

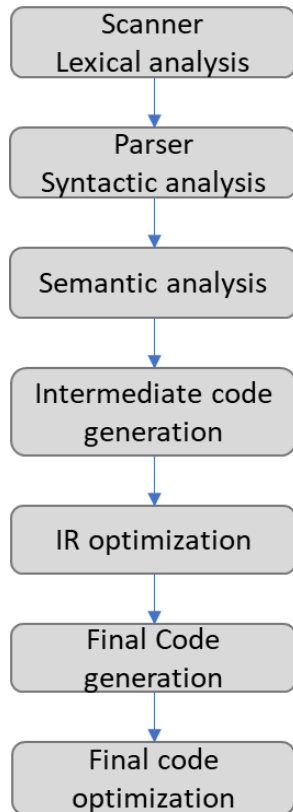
- Convert the character stream from the user into the token stream (the “words” of the programming language)



Last Time

Review: Compiler Overview

Introduced a definition of a compiler and its workflow



Used RegExs to *specify* languages of tokens

Token

Integer Literal

RegEx

$0|(1|2|\dots|9)(0|1|\dots|9)^*$

You Should Know

- The phases of the compiler and what each step does
- What errors the various phases catch
- How to specify a token's lexemes with a regex

This Time

Lecture Outline – Implementing Scanners

From lexical definition to lexical recognition

Why the Regex Makes a Good Lexical Specification

Lecture Outline – Implementing Scanners

Closed under...

- Concatenation
- Union
- Repetition
- Complementation
- ...and more!

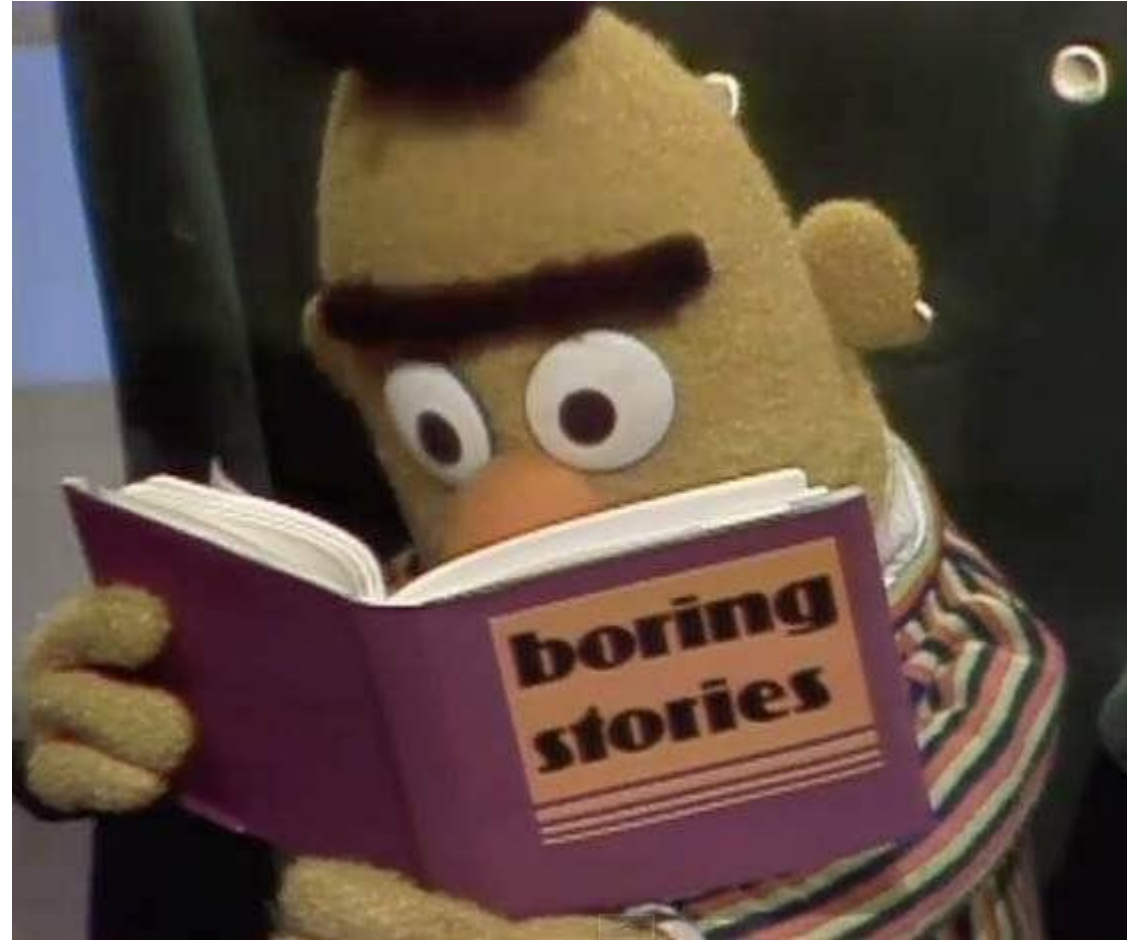
Equivalent to DFAs

By Construction!



RegEx $\equiv$ DFA... So What?

RegEx Properties

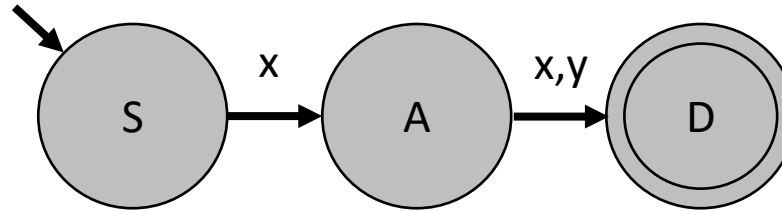


This matters for implementation!

DFA: Easy to Implement

RegEx Properties

DFA (Graph Depiction)



DFA (Successor Notation)

<Current state>,<Transition symbol> = <next state>

$S, x = A$

$A, x = D$

$A, y = D$

DFA (Array Implementation)

Row: current state

Column: transition symbol

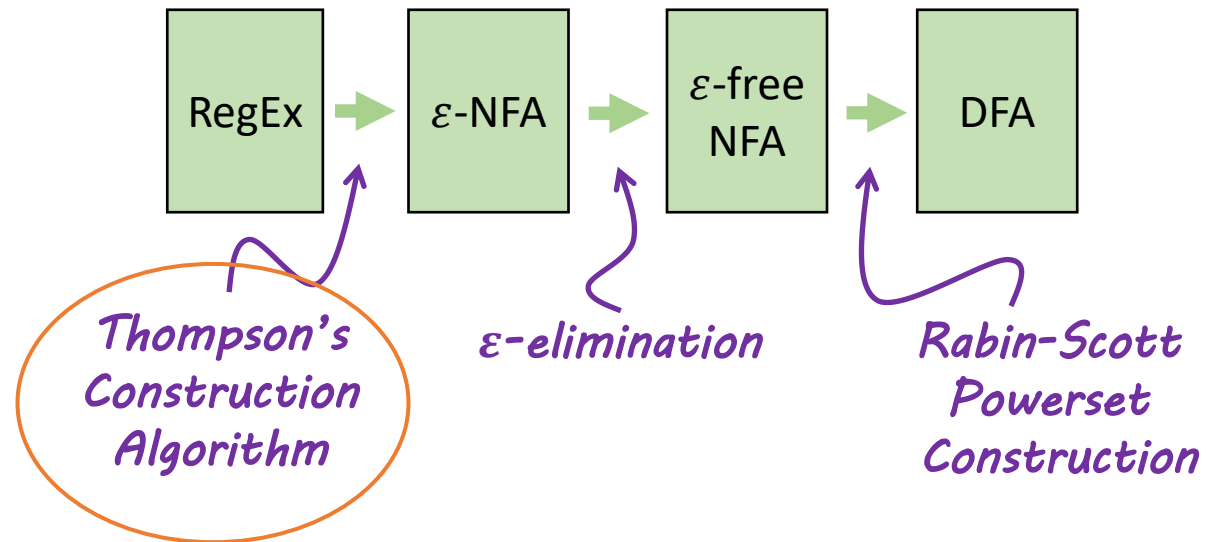
Cell: next state

	x	y
S	A	
A	D	D
D		

Lecture Overview

Lecture 2 – Implementing Scanners

Walk through the translation process formally



Key Concept

Thompson's Construction Algorithm

Use an *expression tree*:

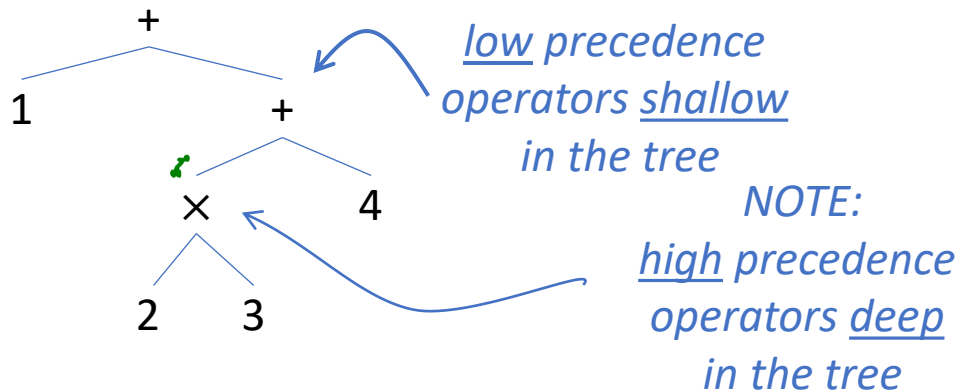
- Leaf: atomic operand
- Branch: operations joining subtrees

Expression Tree Examples

Arithmetic Expression

$1 + 2 \times 3 + 4$

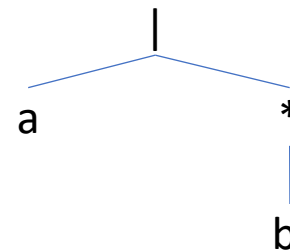
Arithmetic Expression Tree



RegEx

$a|b^*$

Expression Tree



Thompson's Construction Intuition

Thompson's Construction Algorithm

Two-Step Process:

- Break the RegEx down to the simplest units with “obvious” FSMs (i.e. expression tree leaves)
- Combine the sub-FSMs according to operator rules (i.e. expression tree branch rules)

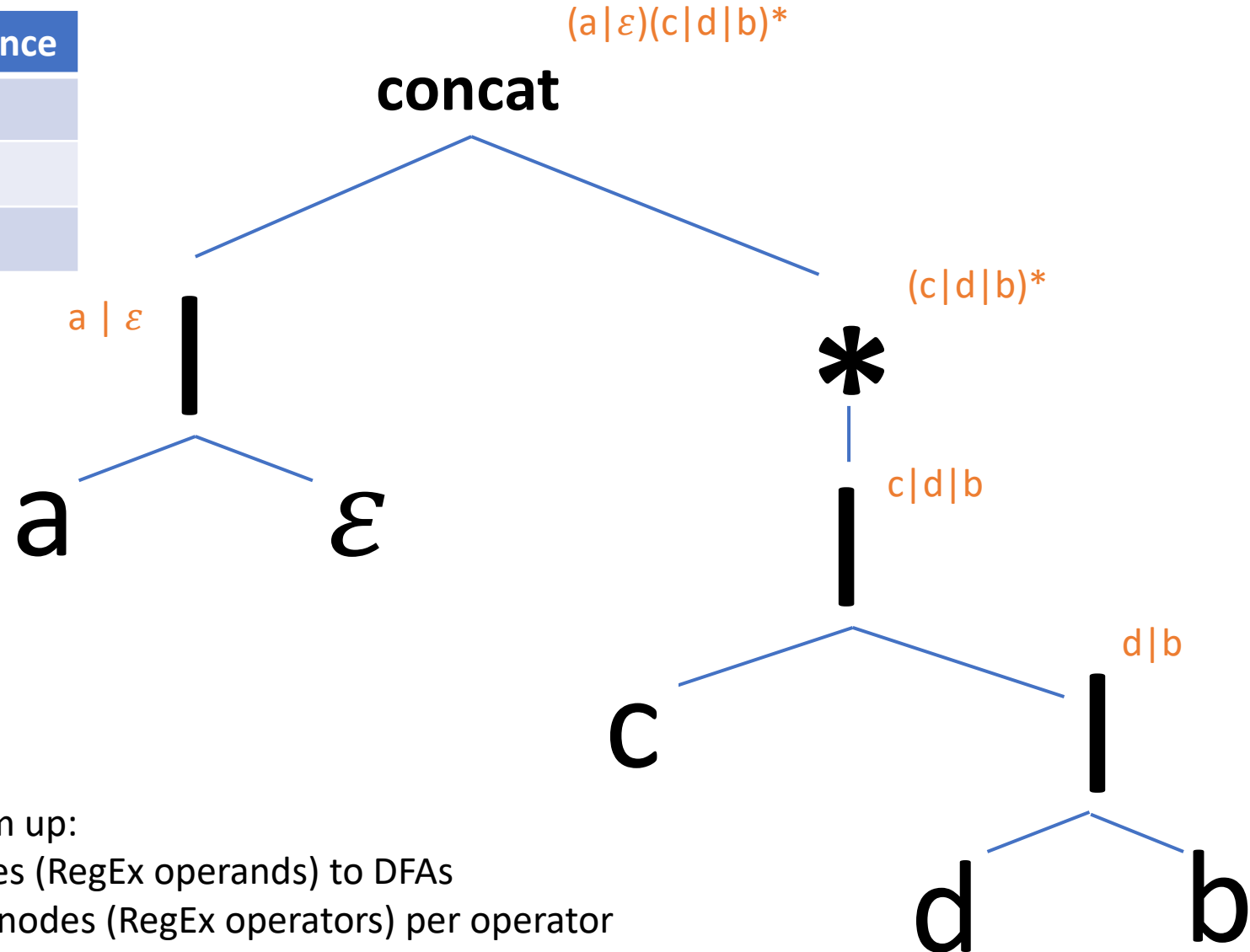


Recombobulate the Regex into an FSM piecewise

Thompson's Construction Alg.

Build the RegEx Tree | Replace nodes bottom-up

Op	Precedence
*	High
concat	Medium
	Low



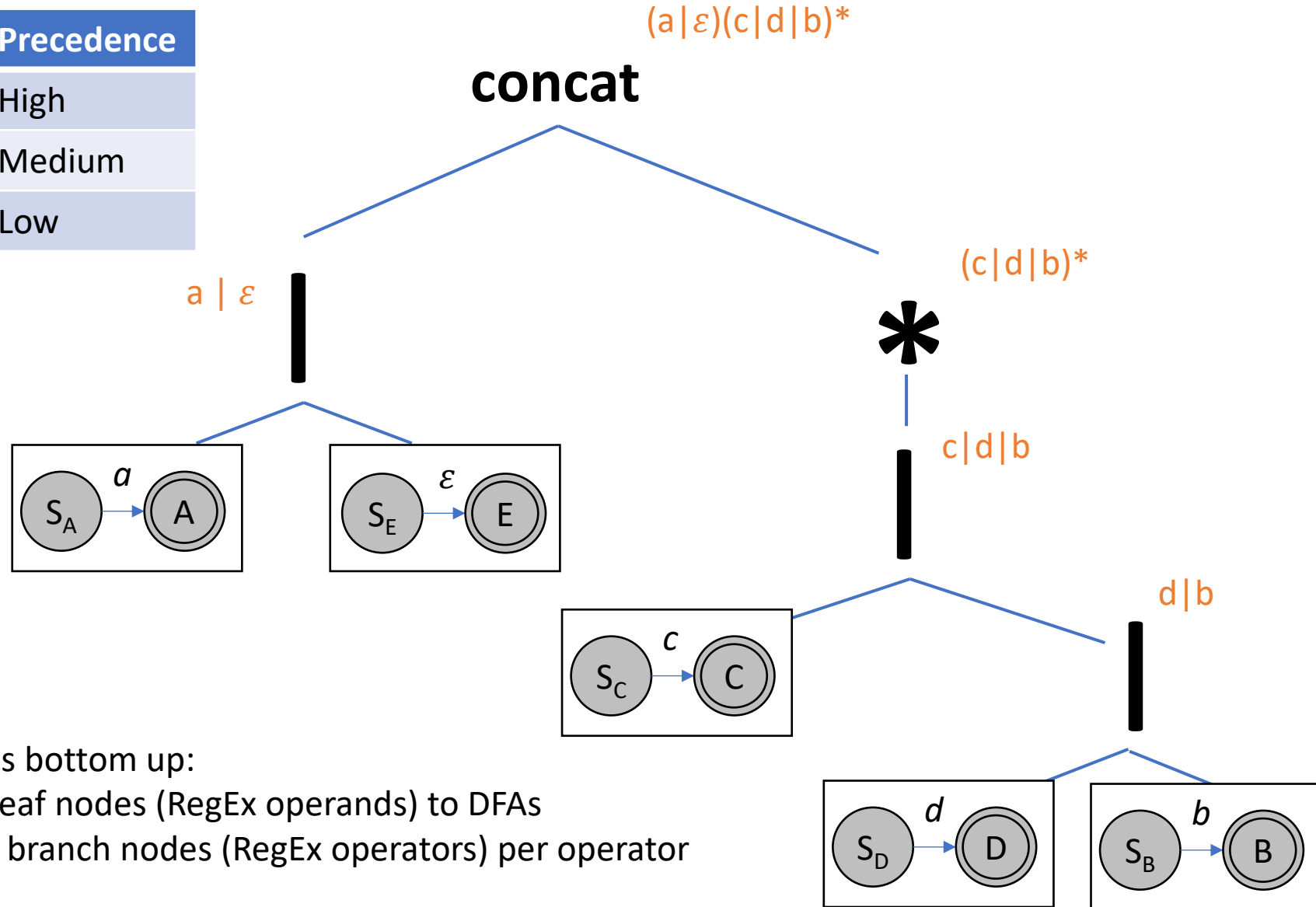
Apply rules bottom up:

- convert leaf nodes (RegEx operands) to DFAs
- combine branch nodes (RegEx operators) per operator

Thompson's Construction Alg.

Build the RegEx Tree | Replace nodes bottom-up

Op	Precedence
*	High
concat	Medium
	Low



Apply rules bottom up:

- convert leaf nodes (RegEx operands) to DFAs
- combine branch nodes (RegEx operators) per operator

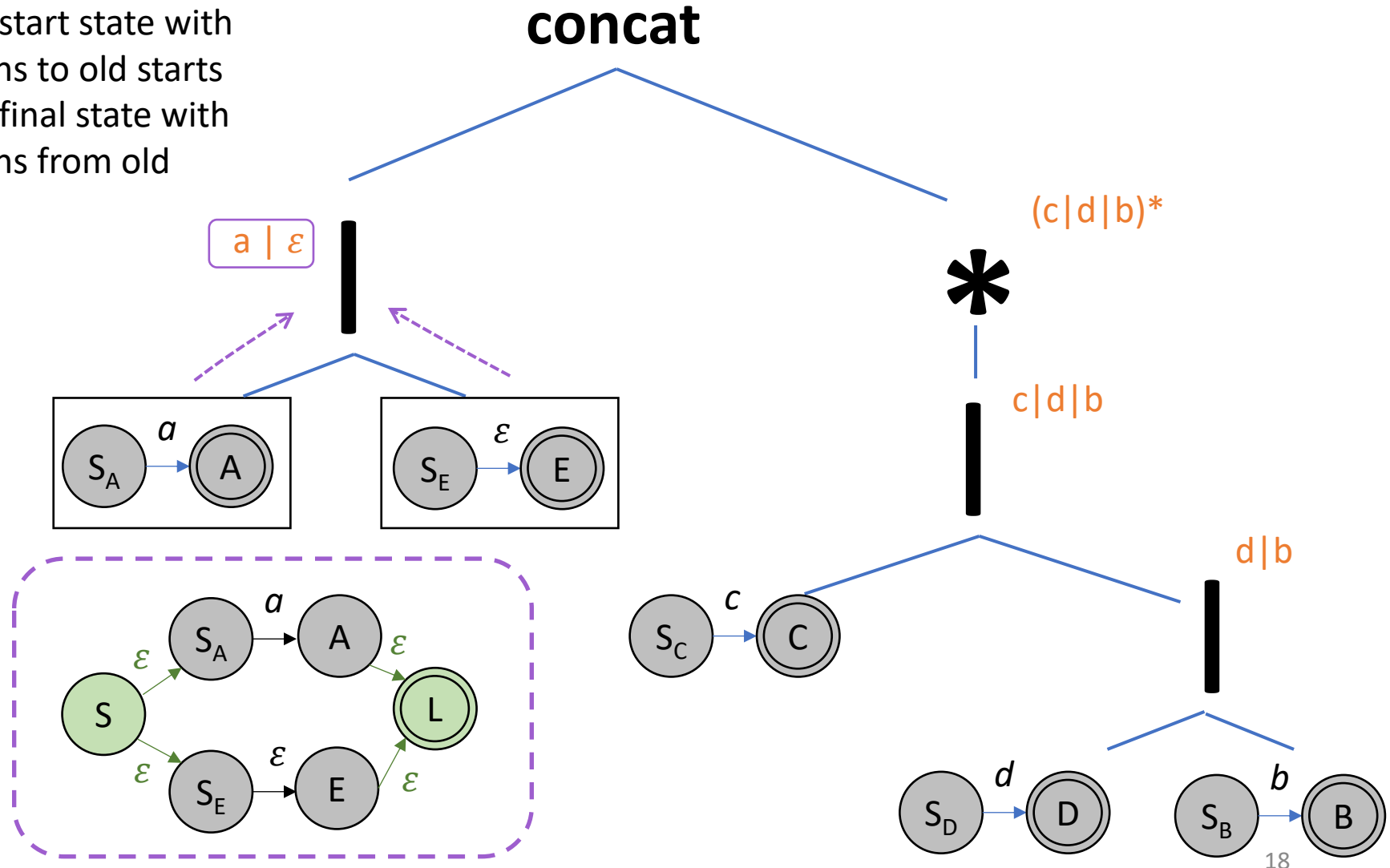
Thompson's Construction Alg.

Build the RegEx Tree Replace nodes bottom-up

$(a|\epsilon)(c|d|b)^*$

Alternation:

- New start state with ϵ -trans to old starts
- New final state with ϵ -trans from old finals



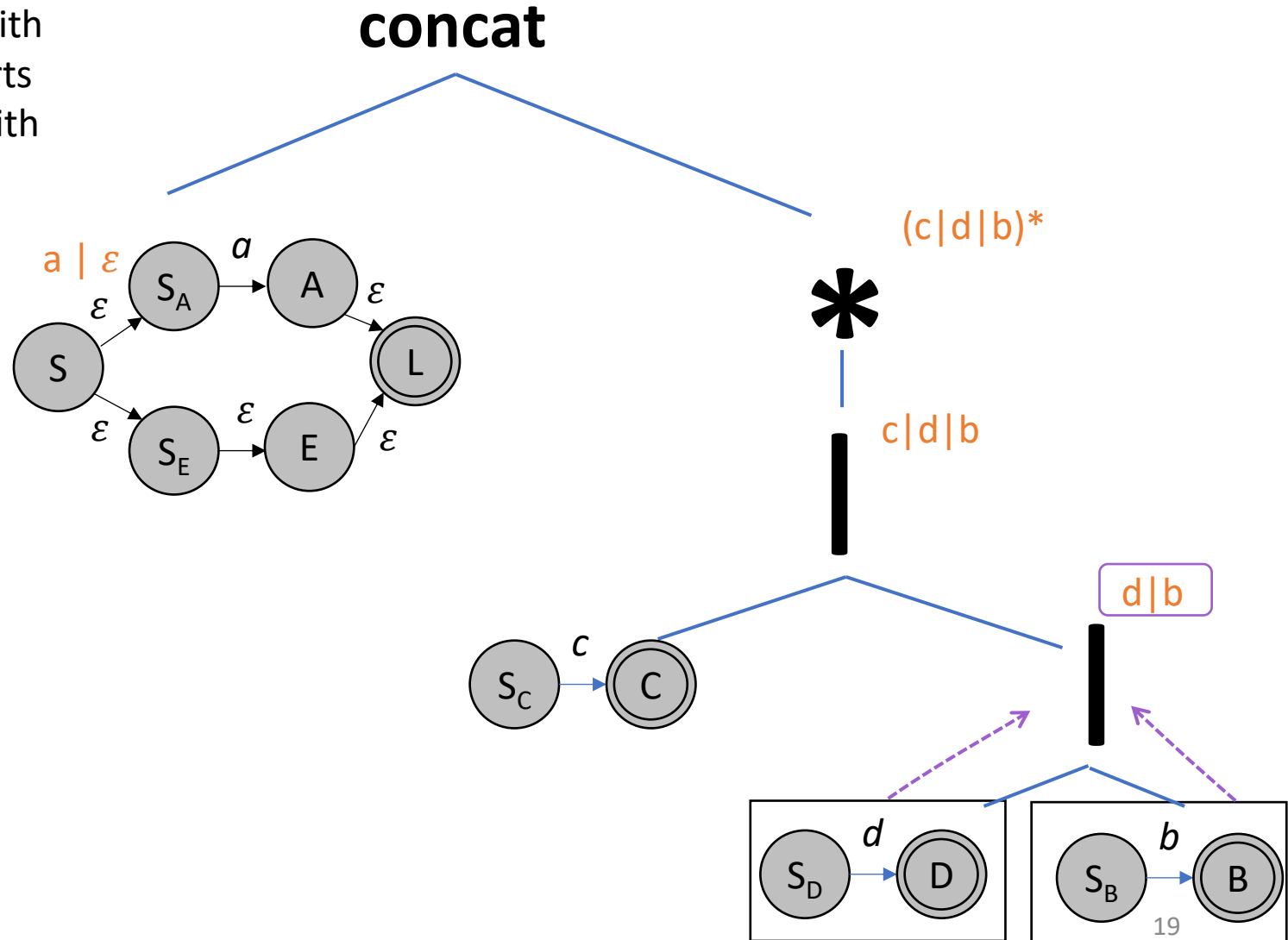
Thompson's Construction Alg.

Build the RegEx Tree Replace nodes bottom-up

$(a|\epsilon)(c|d|b)^*$

Alternation:

- New start state with ϵ -trans to old starts
- New final state with ϵ -trans from old finals



Thompson's Construction Alg.

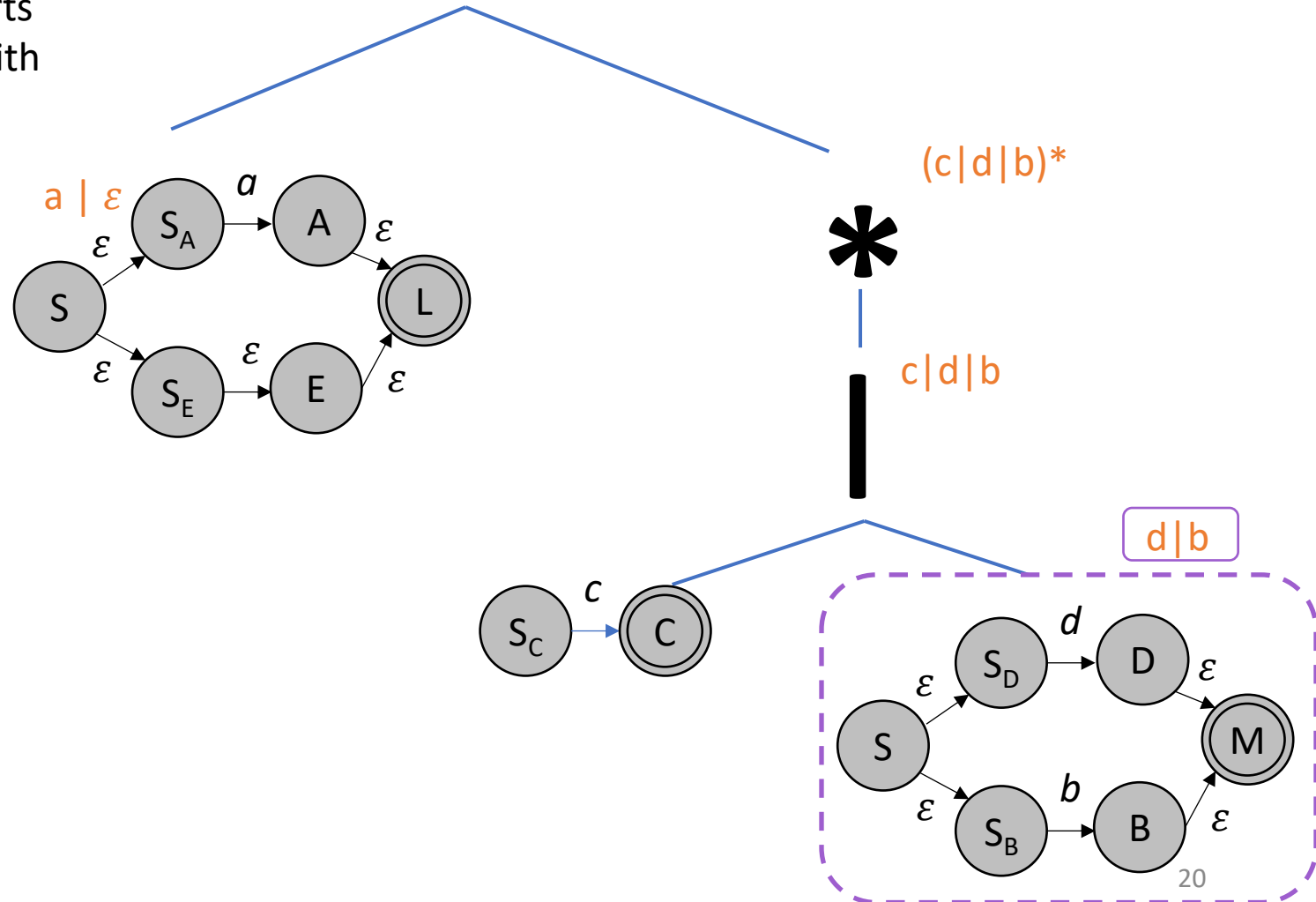
Build the RegEx Tree Replace nodes bottom-up

$(a|\epsilon)(c|d|b)^*$

Alternation:

- New start state with ϵ -trans to old starts
- New final state with ϵ -trans from old finals

concat



Thompson's Construction Alg.

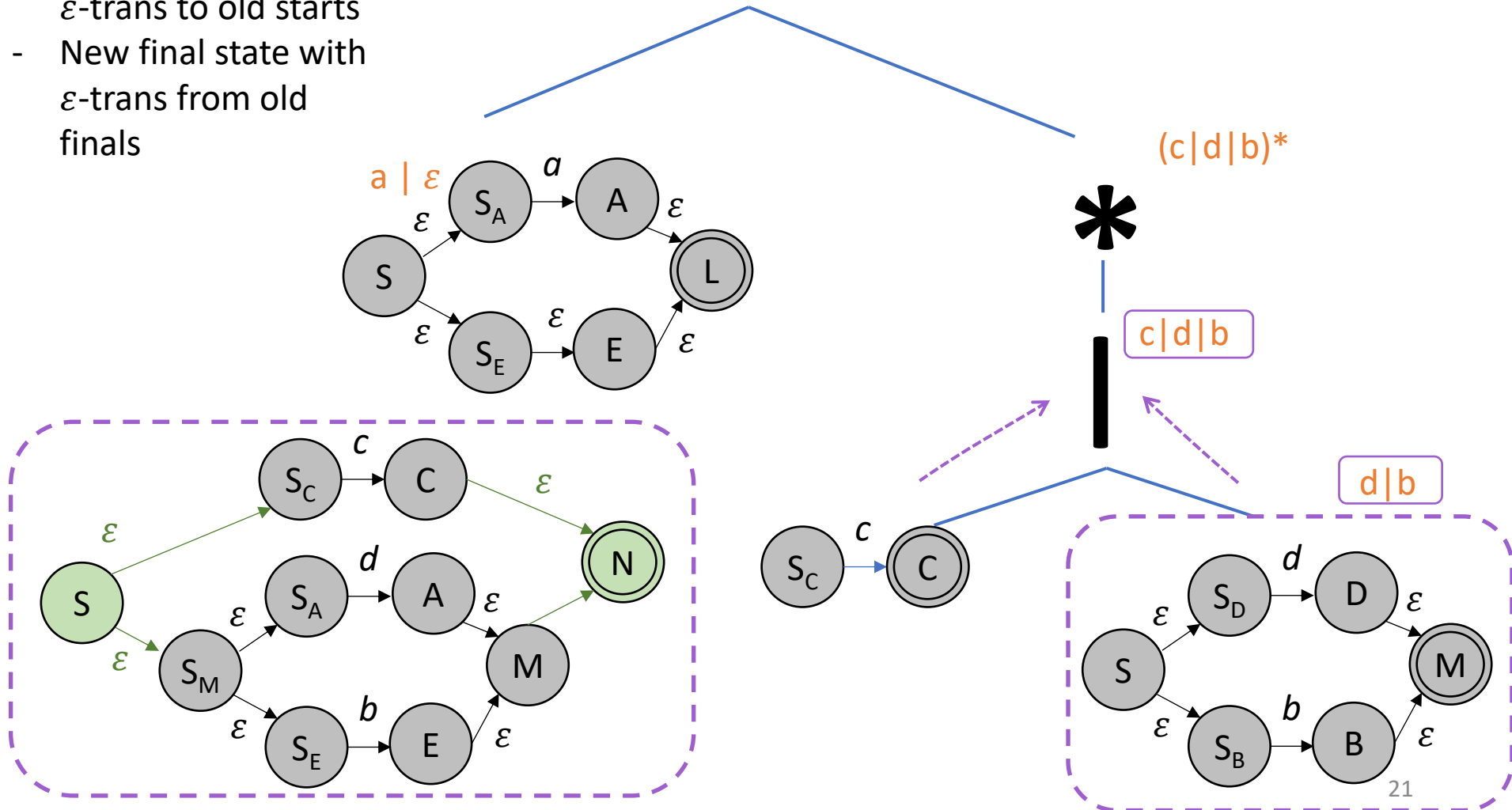
Build the RegEx Tree Replace nodes bottom-up

$(a|\epsilon)(c|d|b)^*$

Alternation:

- New start state with ϵ -trans to old starts
- New final state with ϵ -trans from old finals

concat



Thompson's Construction Alg.

Build the RegEx Tree

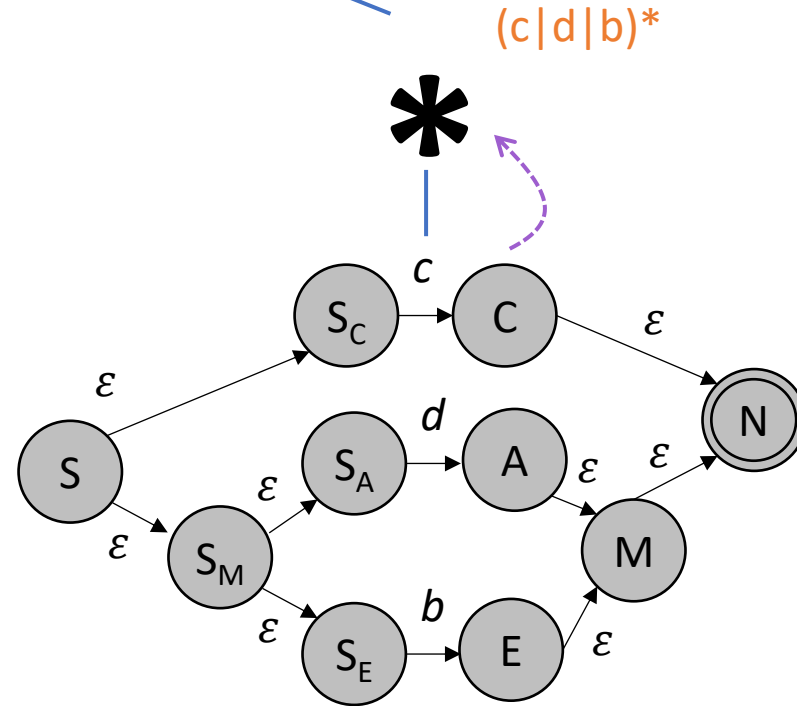
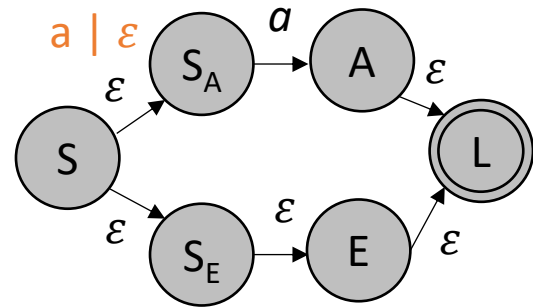
Replace nodes bottom-up

$(a|\epsilon)(c|d|b)^*$

Alternation:

- New start state with ϵ -trans to old starts
- New final state with ϵ -trans from old finals

concat



Thompson's Construction Alg.

Build the RegEx Tree

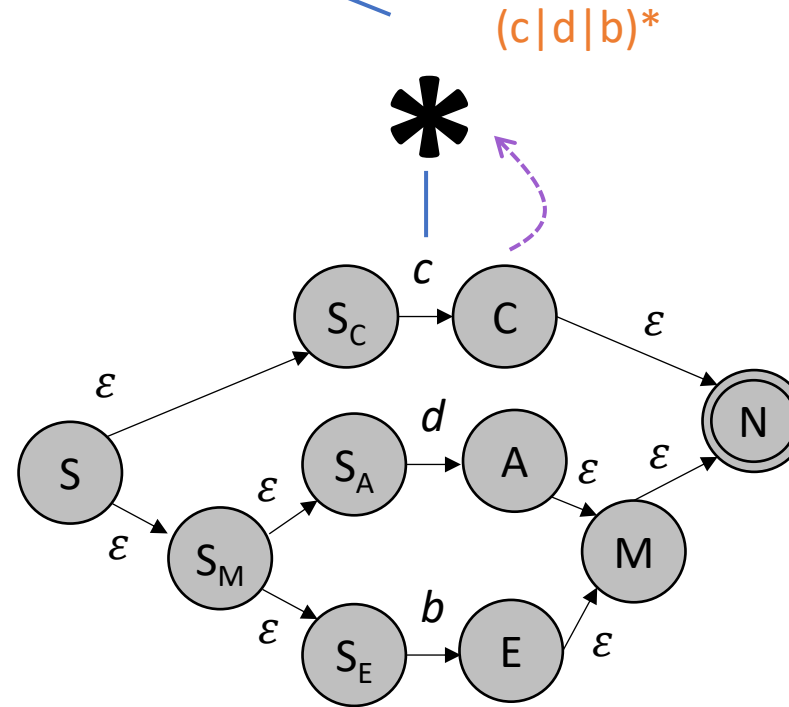
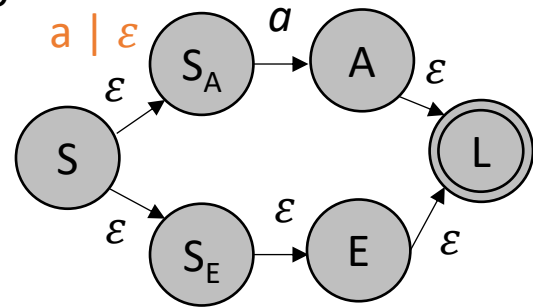
Replace nodes bottom-up

$(a|\epsilon)(c|d|b)^*$

concat

Repetition (* operator):

- New start state with ϵ -edge to old start
- New final state with ϵ -edge from new start
- ϵ -edge from final to start

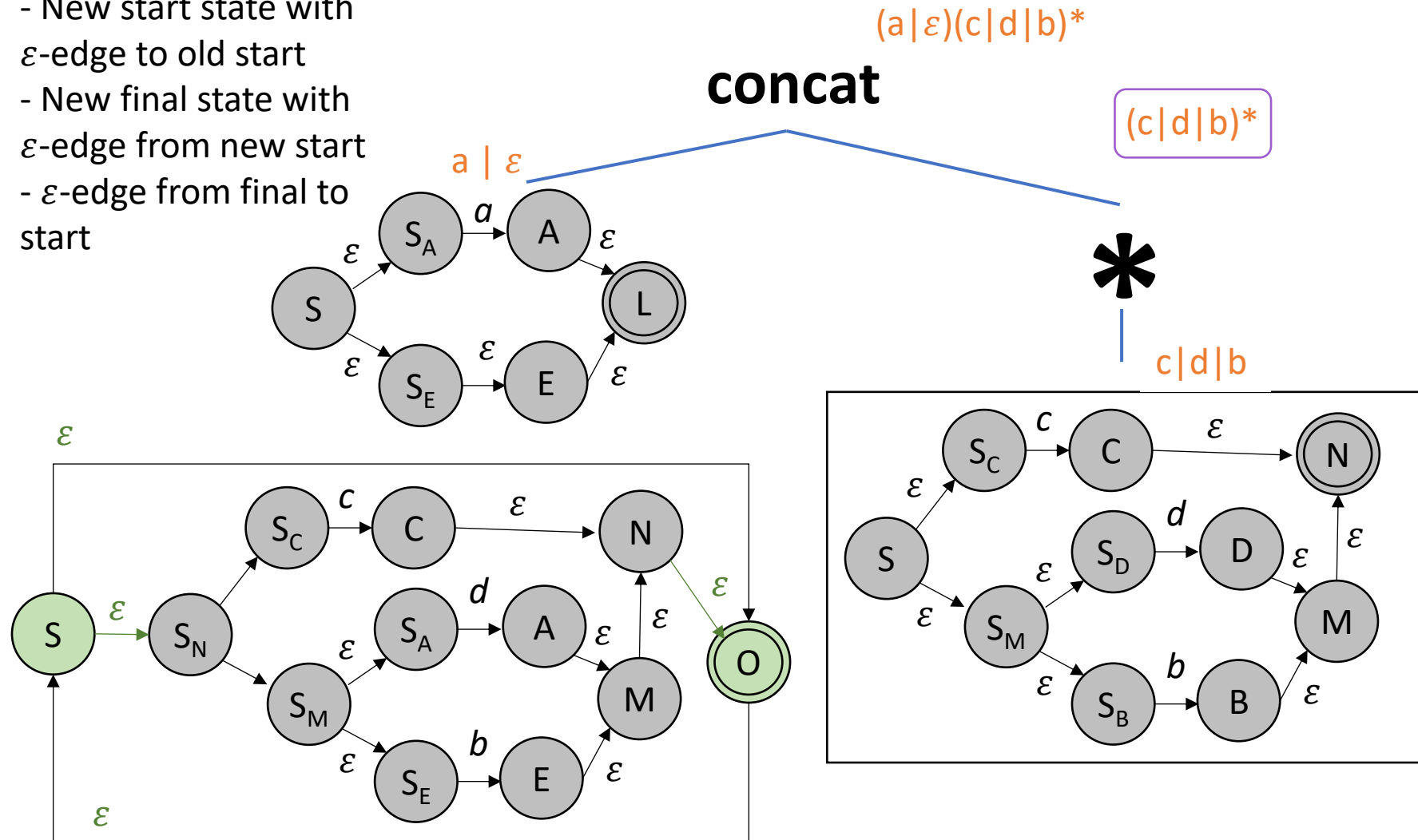


Thompson's Construction Alg.

Build the RegEx Tree Replace nodes bottom-up

Repetition (* operator):

- New start state with ϵ -edge to old start
- New final state with ϵ -edge from new start
- ϵ -edge from final to start



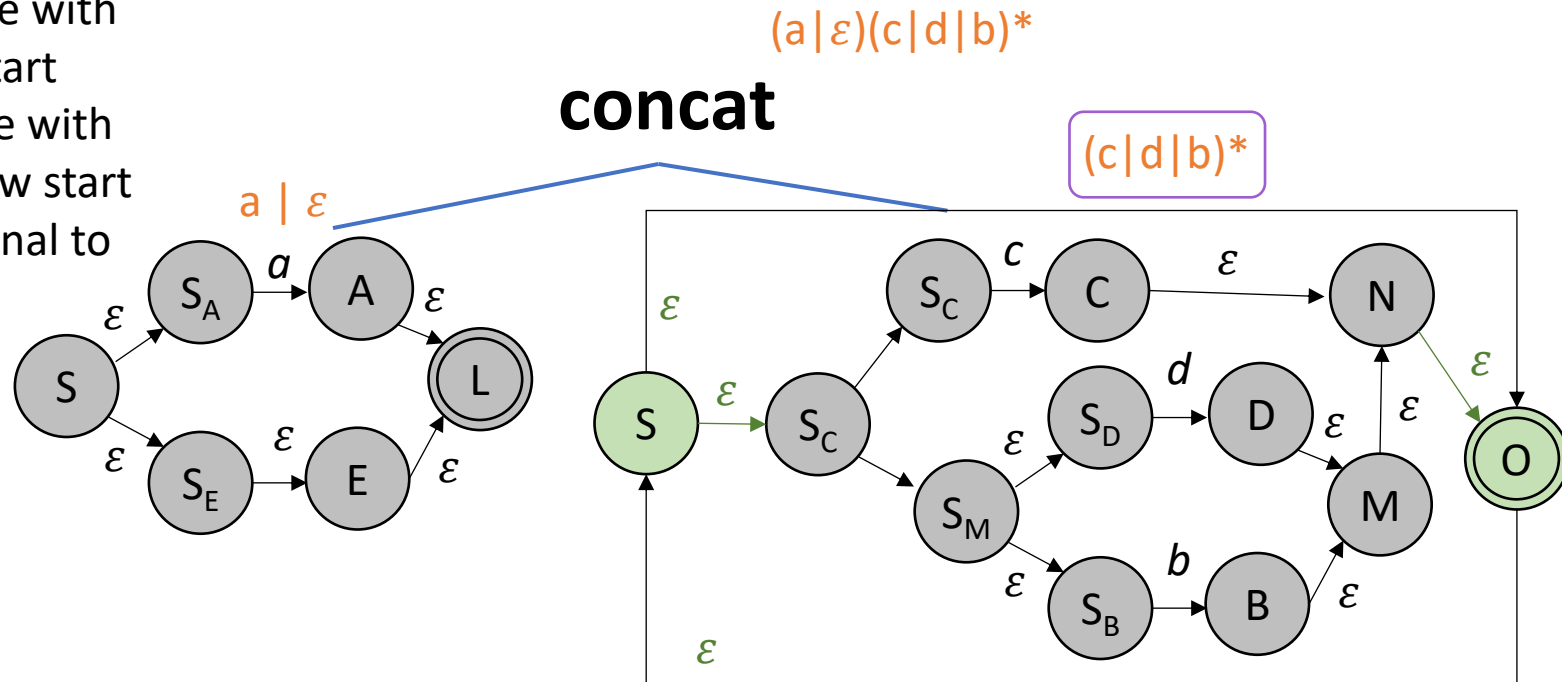
Thompson's Construction Alg.

Build the RegEx Tree

Replace nodes bottom-up

Repetition (* operator):

- New start state with ϵ -edge to old start
- New final state with ϵ -edge from new start
- ϵ -edge from final to start



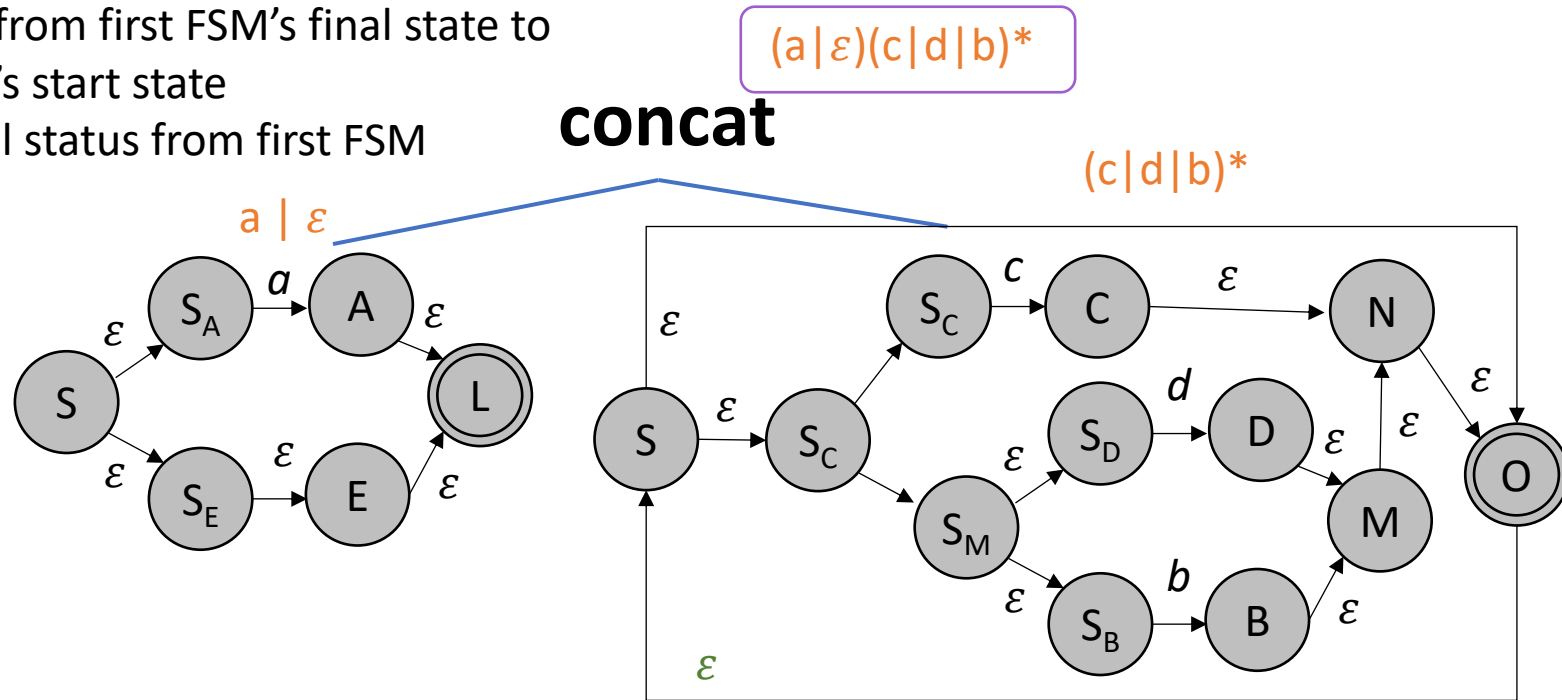
Thompson's Construction Alg.

Build the RegEx Tree

Replace nodes bottom-up

Concatenation:

- Add ϵ -edge from first FSM's final state to second FSM's start state
- Remove final status from first FSM



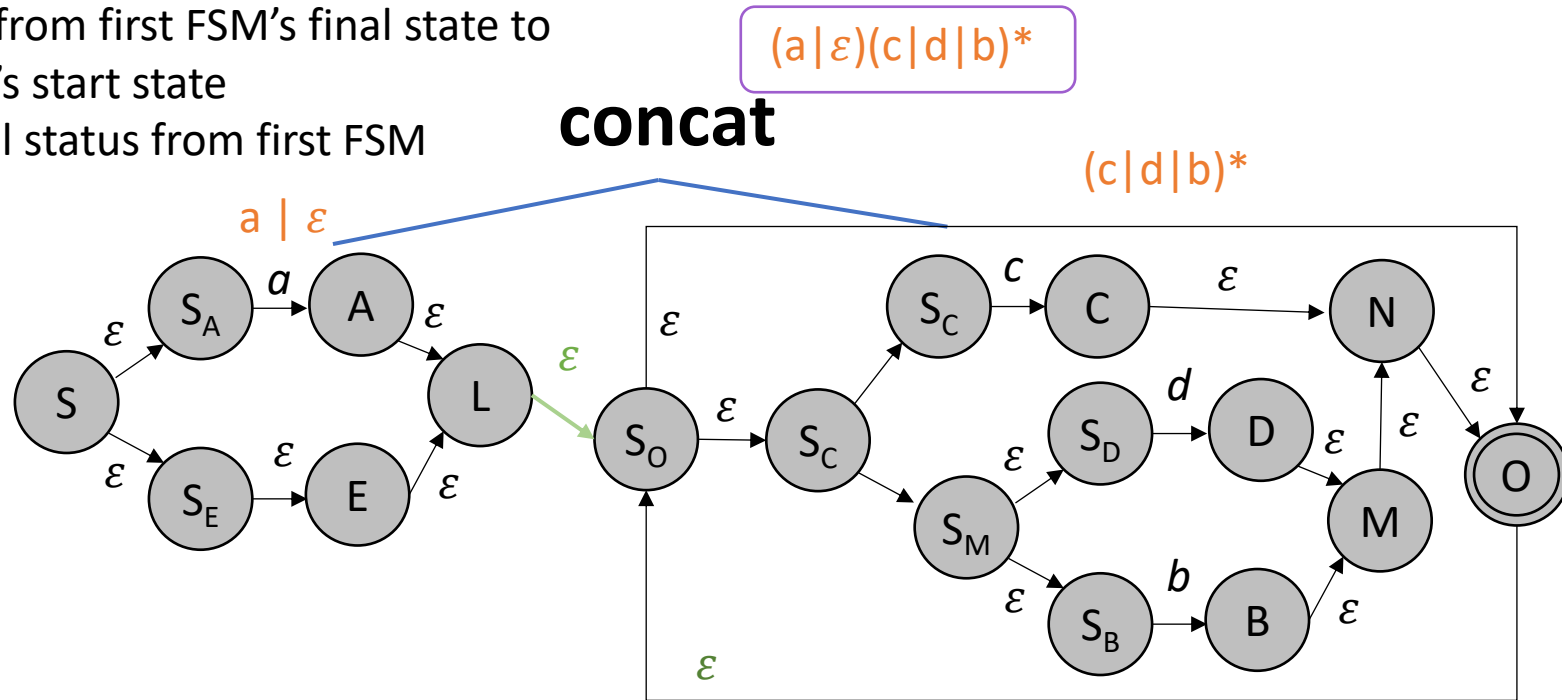
Thompson's Construction Alg.

Build the RegEx Tree

Replace nodes bottom-up

Concatenation:

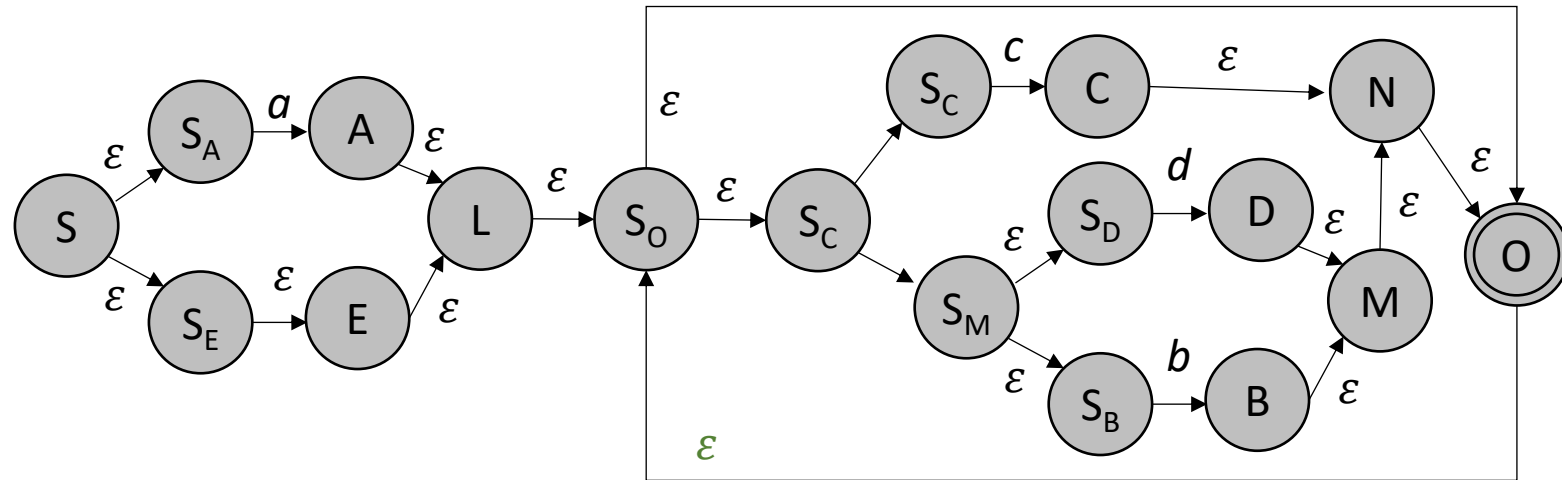
- Add ϵ -edge from first FSM's final state to second FSM's start state
- Remove final status from first FSM



Thompson's Construction Alg.

Build the RegEx Tree | Replace nodes bottom-up

$(a|\epsilon)(c|d|b)^*$

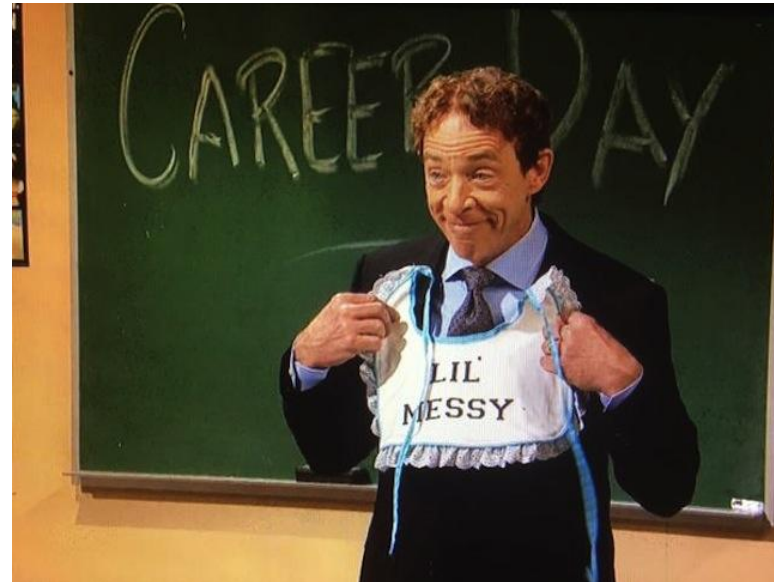


Thompson's Construction: Side-Note

Build the RegEx Tree | Replace nodes bottom-up

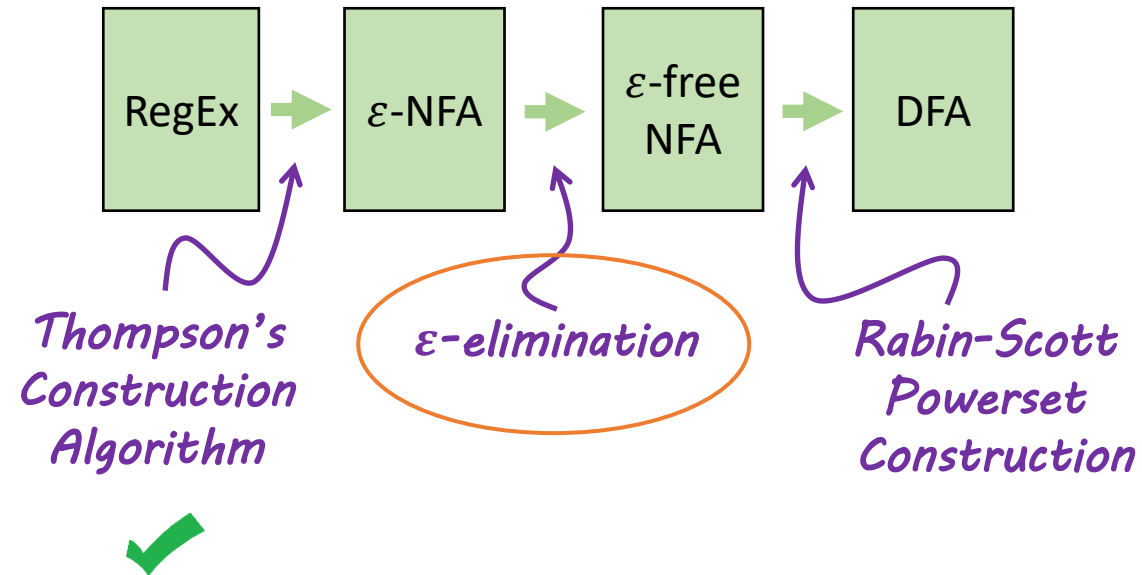
The FSMs produced by Thompsons Construction are a little bit messy!

- Clearly less efficient than what we would do by hand
- Designed for ease of proofs
- In practice, it's easy to minimize FSMs later



From RegEx to DFA

Lecture 2 – Implementing Scanners



Eliminating ϵ -transitions

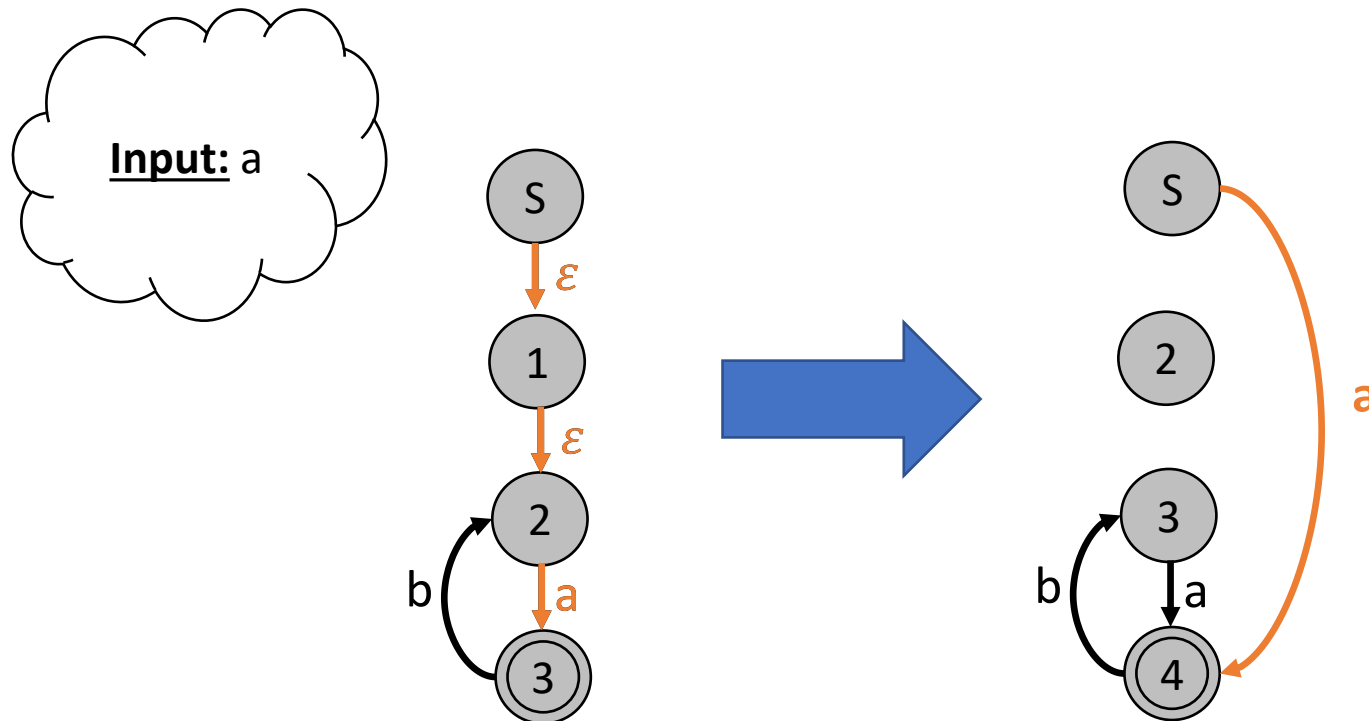
Lecture 2 – Implementing Scanners

Observation: You never see an epsilon in the input

- Consuming a character means taking a “chain” of zero-or-more ϵ -edges then a real character edge

Algorithm Intuition: *cut out the middleman*

- Replace all “chains” with a direct real-character edge



Eliminating ε -transitions

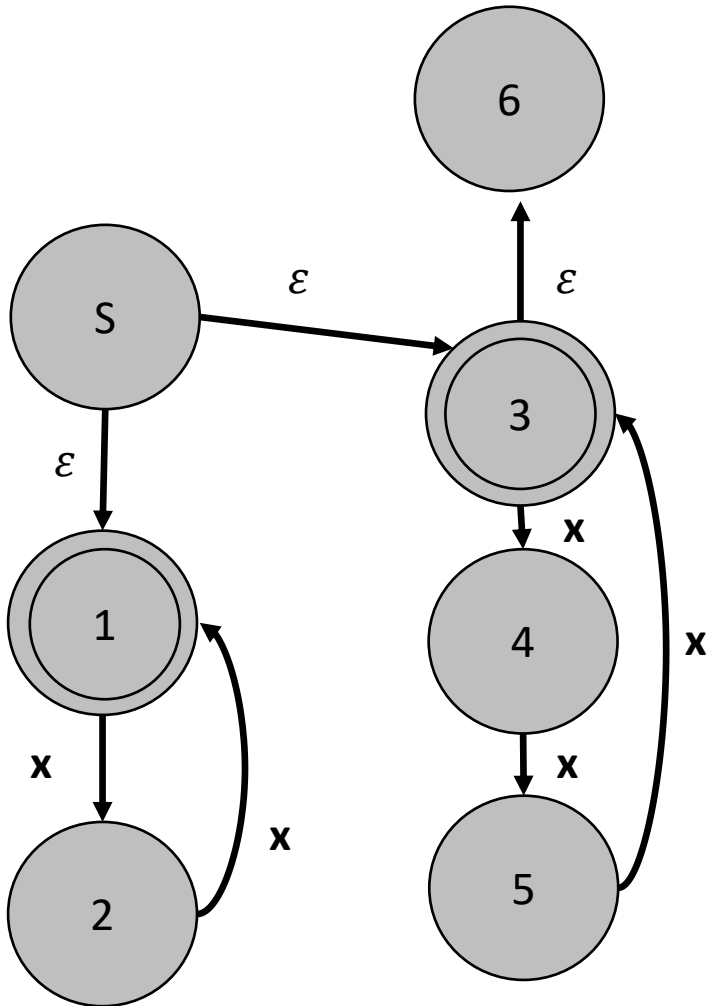
Lecture 2 – Implementing Scanners

- Compute $\varepsilon\text{-close}(s)$, the set of states reachable via 0 or more ε -edges from s
- Copy all states from N to an ε -free version, N'
- Put s in F' if $\varepsilon\text{-close}(s)$ contains a state in F
- Put $s, c \rightarrow t$ in δ' if there is a c -edge to t in $\varepsilon\text{-close}(s)$

Example, Step I

Eliminating ε -Transitions

Let ε -close(s) be the set of states reachable via 0 or more ε -edges



$$\varepsilon\text{-close}(S) = \{S, 1, 3, 6\}$$

$$\varepsilon\text{-close}(3) = \{3, 6\}$$

$$\varepsilon\text{-close}(1) = \{1\}$$

$$\varepsilon\text{-close}(2) = \{2\}$$

$$\varepsilon\text{-close}(4) = \{4\}$$

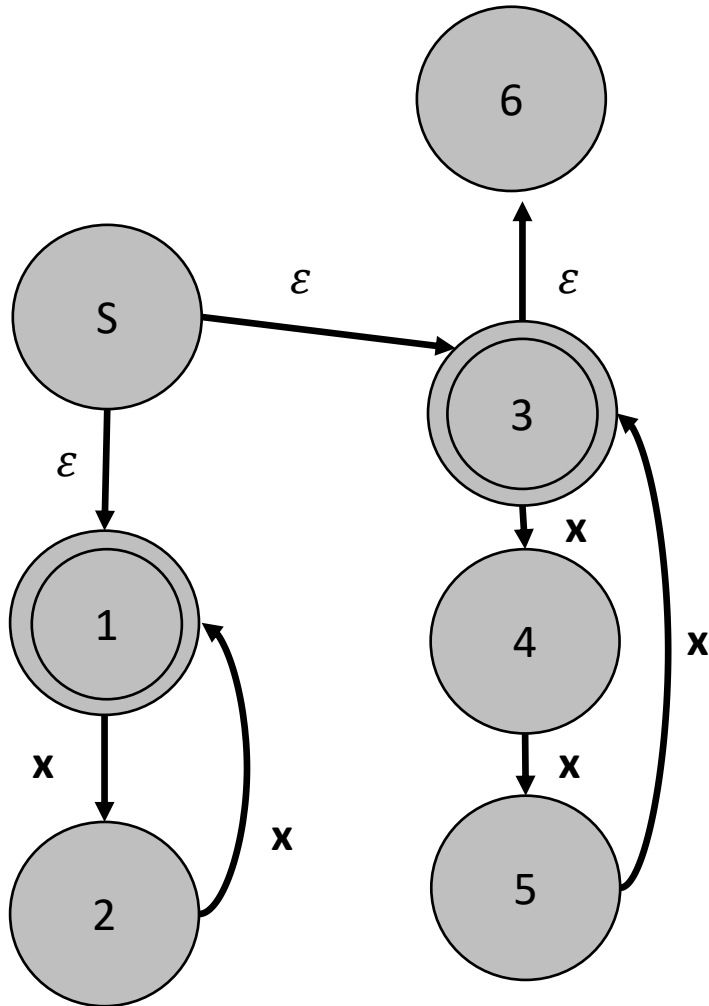
$$\varepsilon\text{-close}(5) = \{5\}$$

$$\varepsilon\text{-close}(6) = \{6\}$$

Example, Step II

Eliminating ε -Transitions

Copy all states from N to N'



$$\varepsilon\text{-close}(S) = \{S, 1, 3, 6\}$$

$$\varepsilon\text{-close}(3) = \{3, 6\}$$

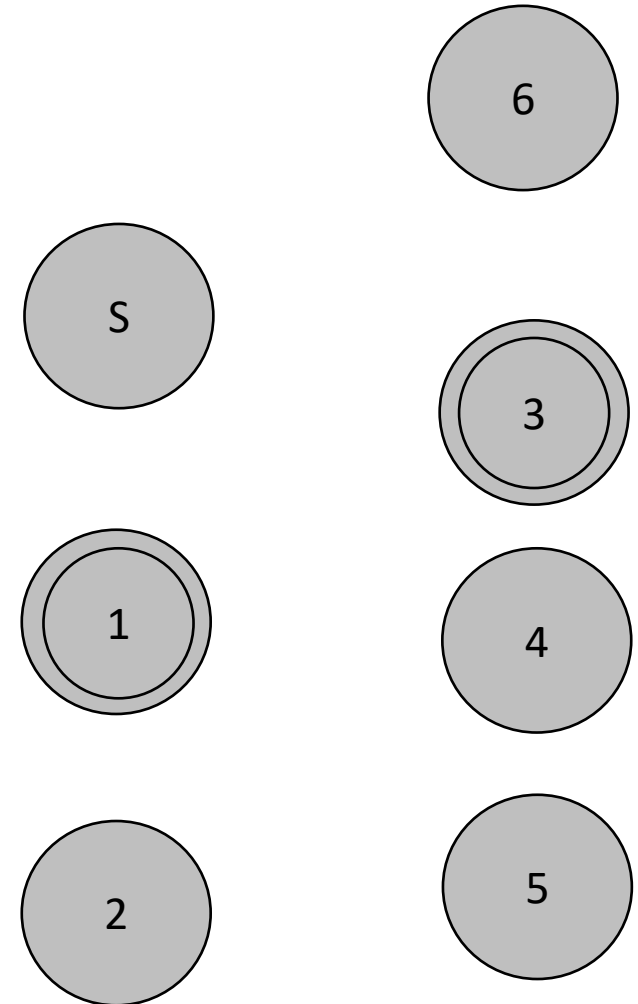
$$\varepsilon\text{-close}(1) = \{1\}$$

$$\varepsilon\text{-close}(2) = \{2\}$$

$$\varepsilon\text{-close}(4) = \{4\}$$

$$\varepsilon\text{-close}(5) = \{5\}$$

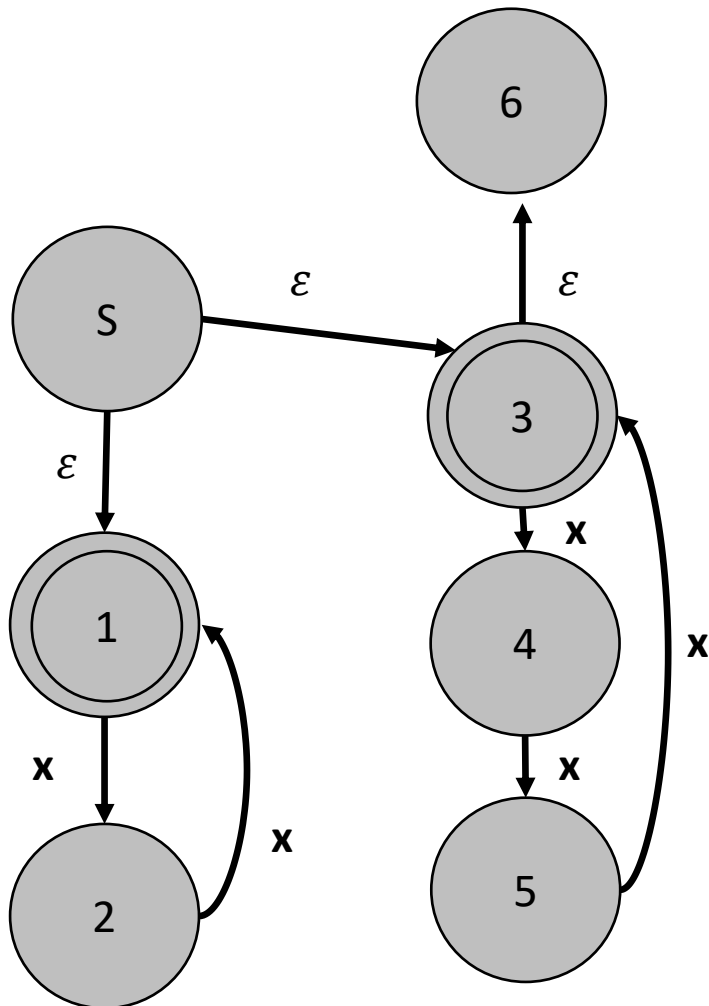
$$\varepsilon\text{-close}(6) = \{6\}$$



Example, Step III

Eliminating ϵ -Transitions

Put s in F' if $\epsilon\text{-close}(s)$ contains a state in F



$$\epsilon\text{-close}(S) = \{S, 1, 3, 6\}$$

$$\epsilon\text{-close}(3) = \{3, 6\}$$

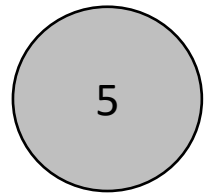
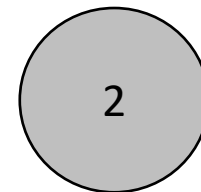
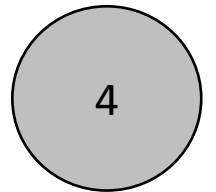
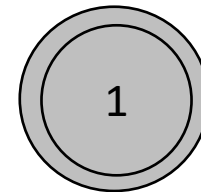
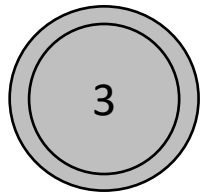
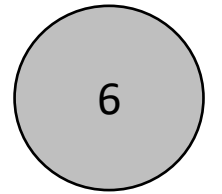
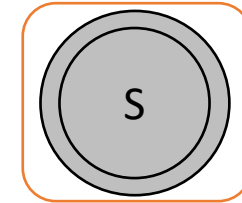
$$\epsilon\text{-close}(1) = \{1\}$$

$$\epsilon\text{-close}(2) = \{2\}$$

$$\epsilon\text{-close}(4) = \{4\}$$

$$\epsilon\text{-close}(5) = \{5\}$$

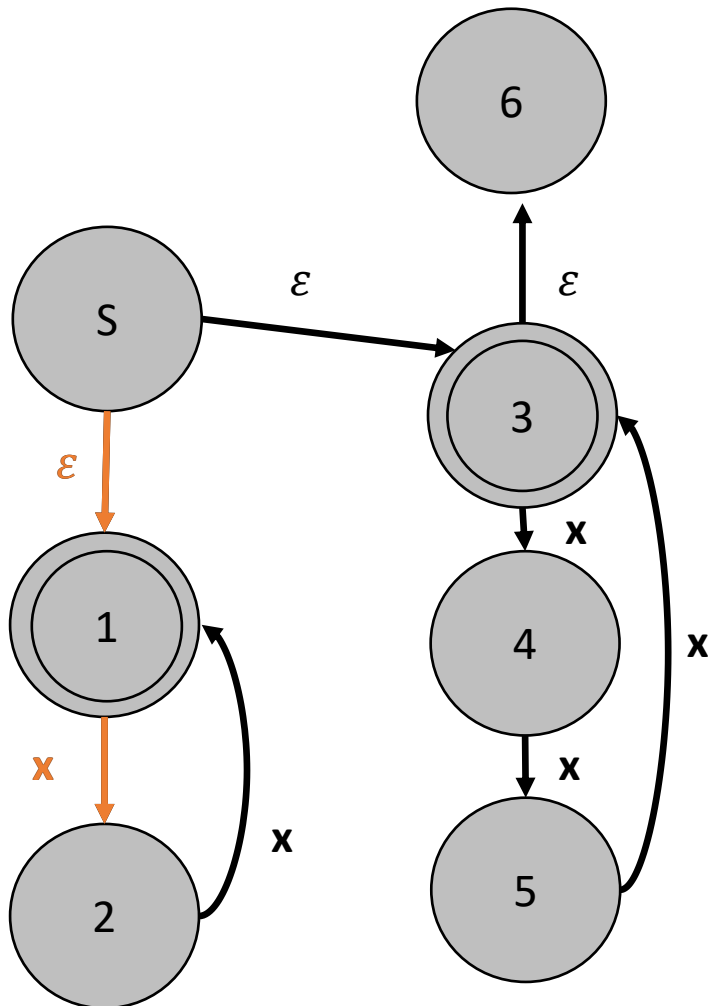
$$\epsilon\text{-close}(6) = \{6\}$$



Example, Step IV

Eliminating ε -Transitions

Put $s, c \rightarrow t$ in δ' if there is a c -edge to t in ε -close(s)



$$\varepsilon\text{-close}(S) = \{S, 1, 3, 6\}$$

$$\varepsilon\text{-close}(3) = \{3, 6\}$$

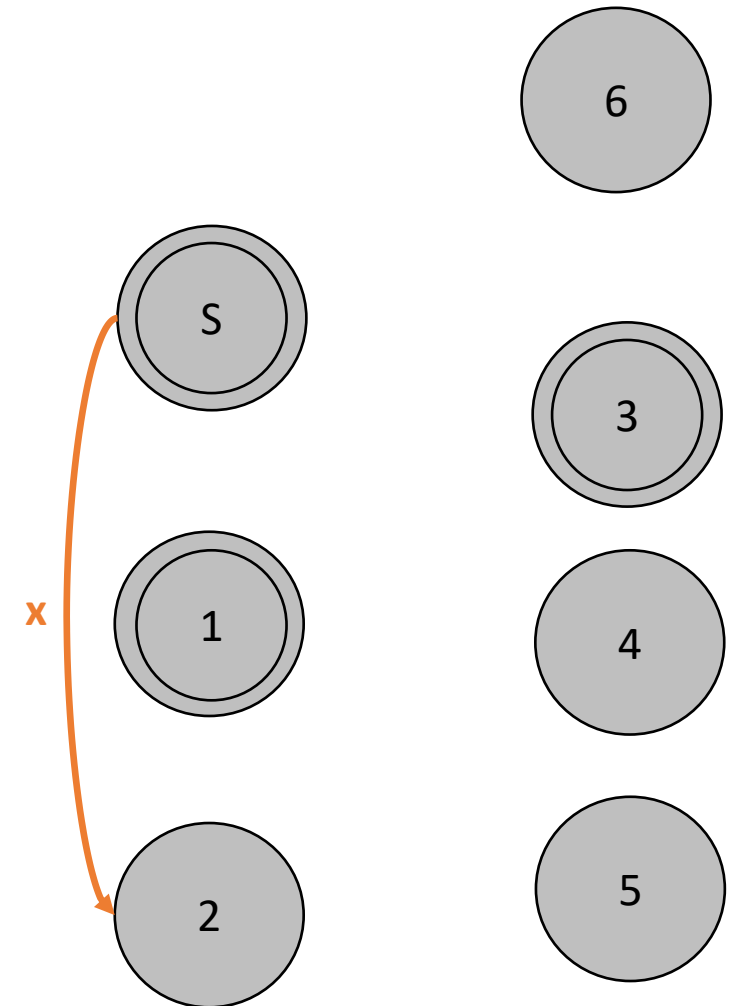
$$\varepsilon\text{-close}(1) = \{1\}$$

$$\varepsilon\text{-close}(2) = \{2\}$$

$$\varepsilon\text{-close}(4) = \{4\}$$

$$\varepsilon\text{-close}(5) = \{5\}$$

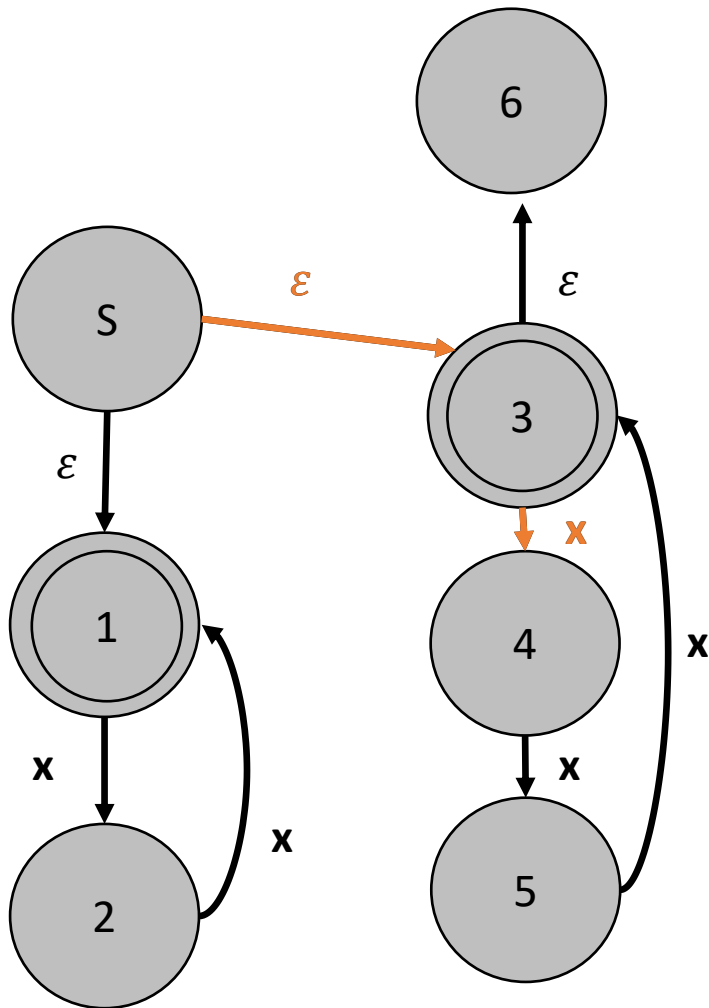
$$\varepsilon\text{-close}(6) = \{6\}$$



Example, Step IV

Eliminating ε -Transitions

Put $s, c \rightarrow t$ in δ' if there is a c -edge to t in $\varepsilon\text{-close}(s)$



$$\varepsilon\text{-close}(S) = \{S, 1, 3, 6\}$$

$$\varepsilon\text{-close}(3) = \{3, 6\}$$

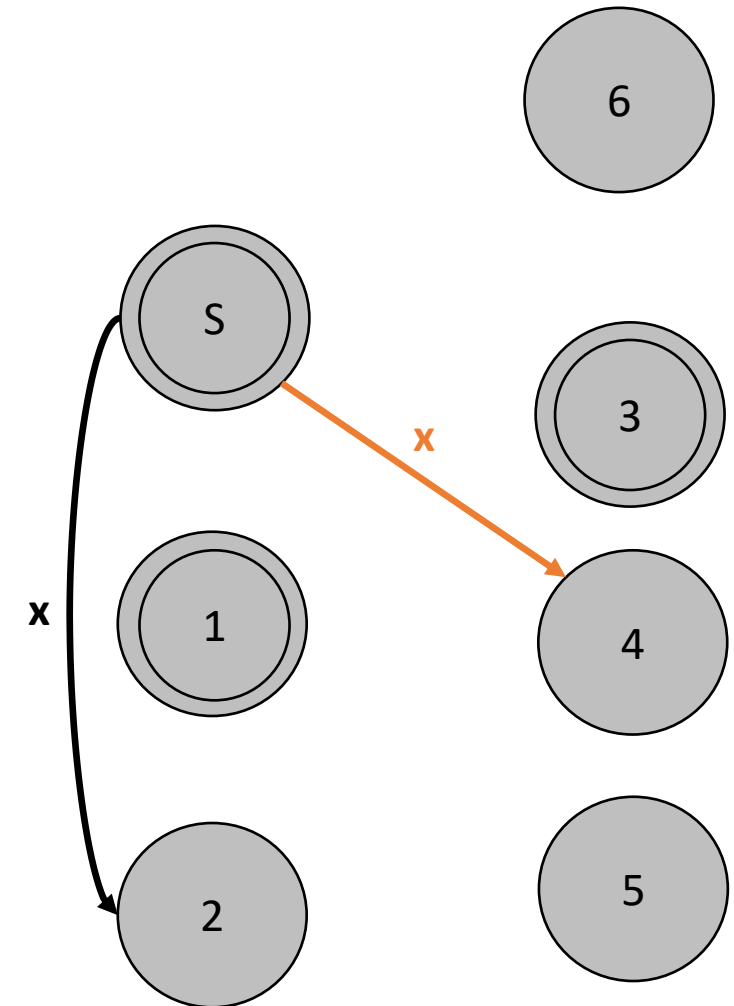
$$\varepsilon\text{-close}(1) = \{1\}$$

$$\varepsilon\text{-close}(2) = \{2\}$$

$$\varepsilon\text{-close}(4) = \{4\}$$

$$\varepsilon\text{-close}(5) = \{5\}$$

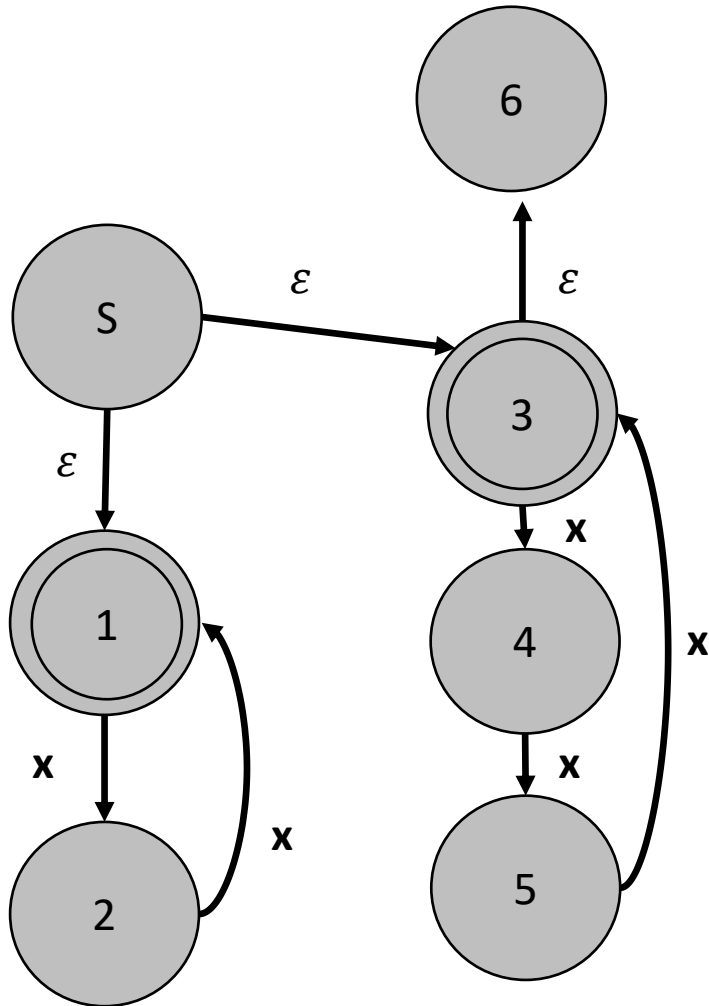
$$\varepsilon\text{-close}(6) = \{6\}$$



Example, Step IV

Eliminating ε -Transitions

Put $s, c \rightarrow t$ in δ' if there is a c -edge to t in ε -close(s)



$$\varepsilon\text{-close}(S) = \{S, 1, 3, 6\}$$

$$\varepsilon\text{-close}(3) = \{3, 6\}$$

$$\varepsilon\text{-close}(1) = \{1\}$$

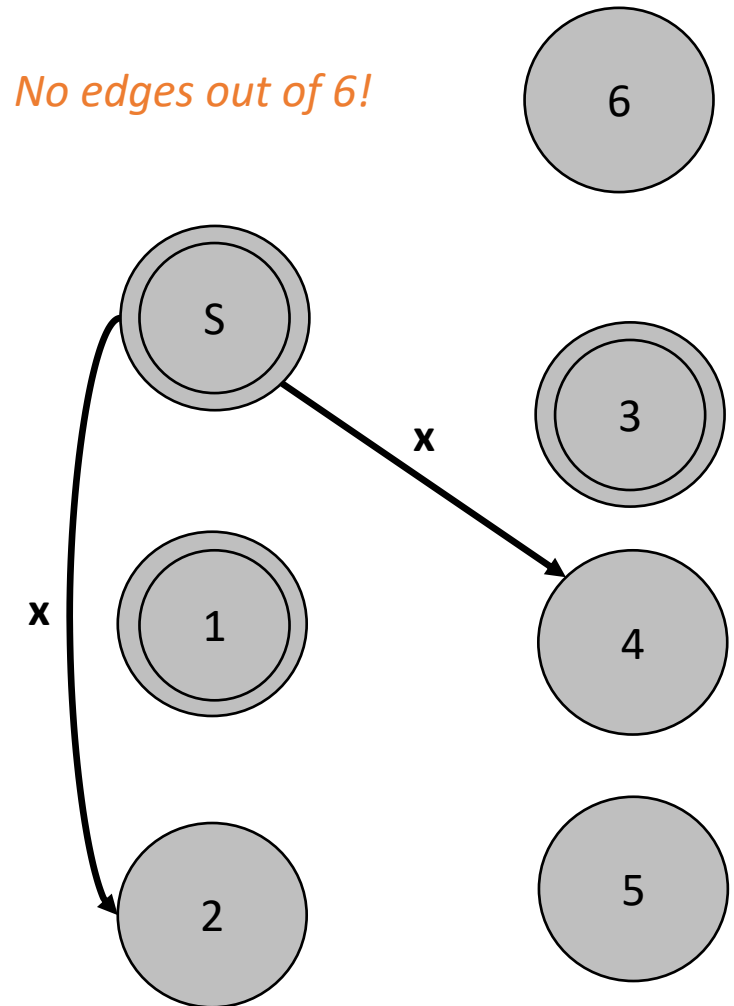
$$\varepsilon\text{-close}(2) = \{2\}$$

$$\varepsilon\text{-close}(4) = \{4\}$$

$$\varepsilon\text{-close}(5) = \{5\}$$

$$\varepsilon\text{-close}(6) = \{6\}$$

No edges out of 6!

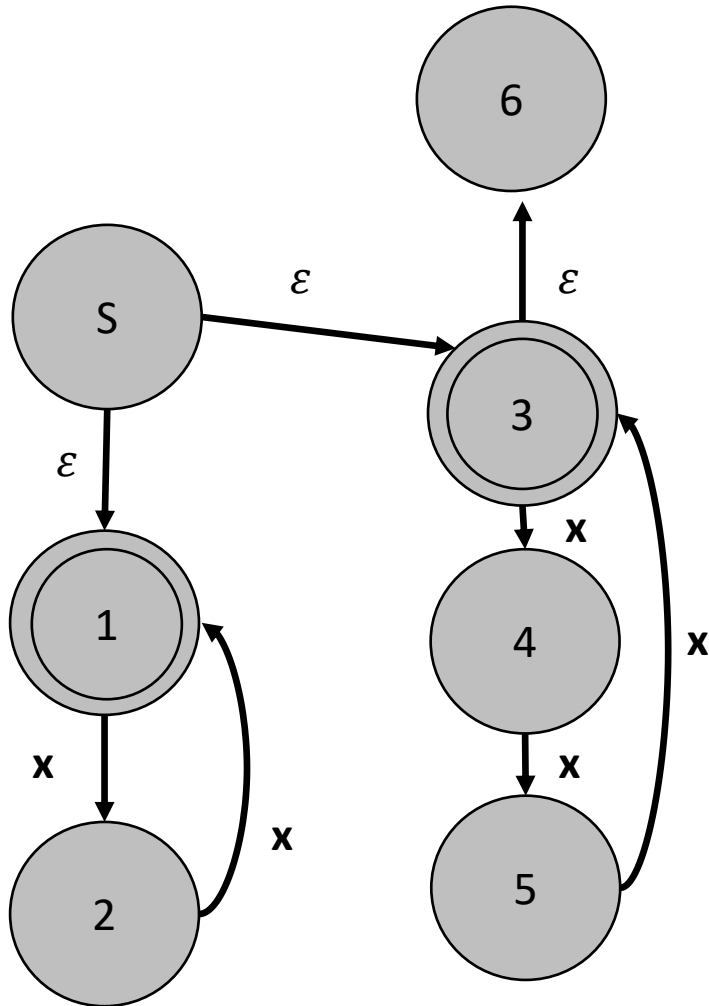


Example, Step IV

Eliminating ε -Transitions

Put $s, c \rightarrow t$ in δ' if there is a c -edge to t in $\varepsilon\text{-close}(s)$

Note: this definition necessarily preserves all original non- ε edges



$$\varepsilon\text{-close}(S) = \{S, 1, 3, 6\}$$

$$\varepsilon\text{-close}(3) = \{3, 6\}$$

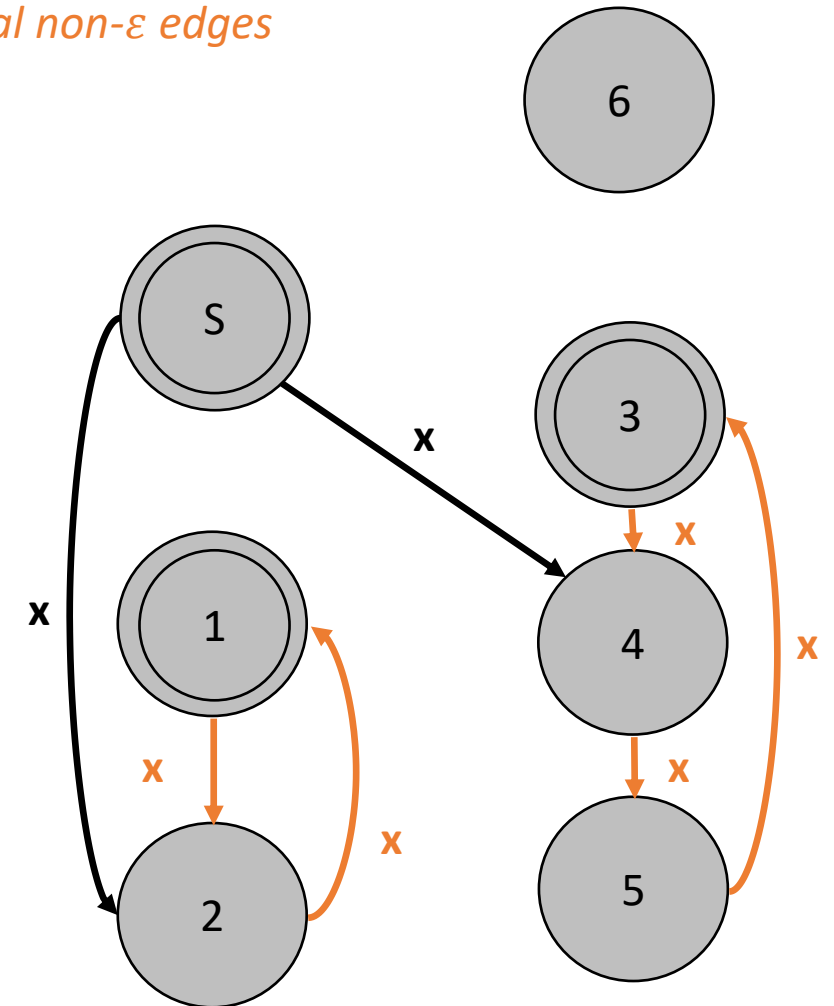
$$\varepsilon\text{-close}(1) = \{1\}$$

$$\varepsilon\text{-close}(2) = \{2\}$$

$$\varepsilon\text{-close}(4) = \{4\}$$

$$\varepsilon\text{-close}(5) = \{5\}$$

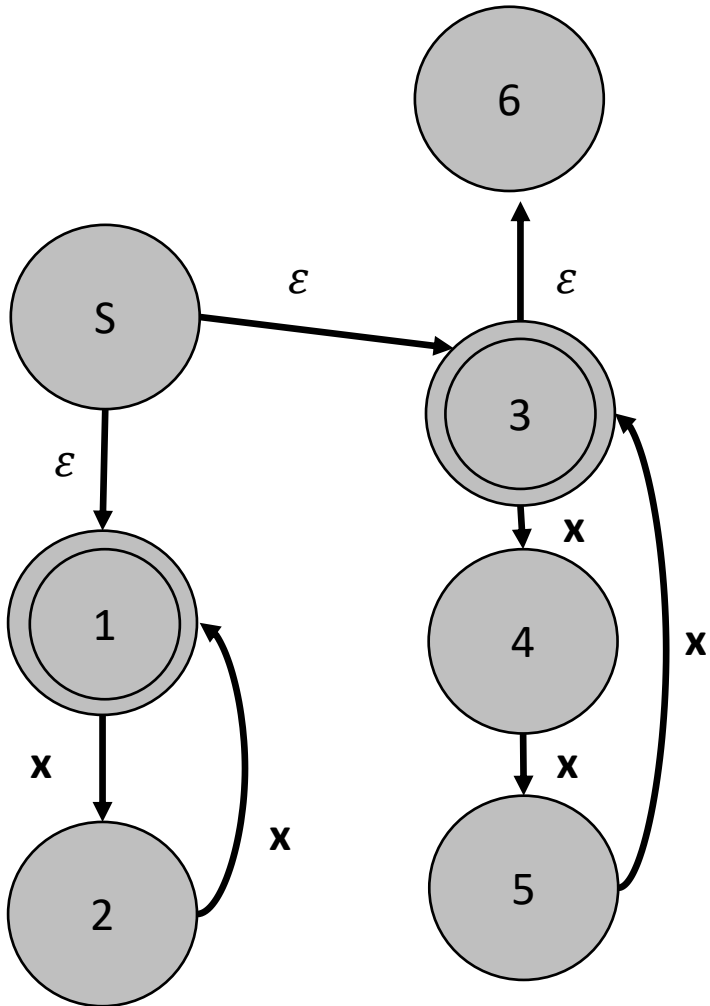
$$\varepsilon\text{-close}(6) = \{6\}$$



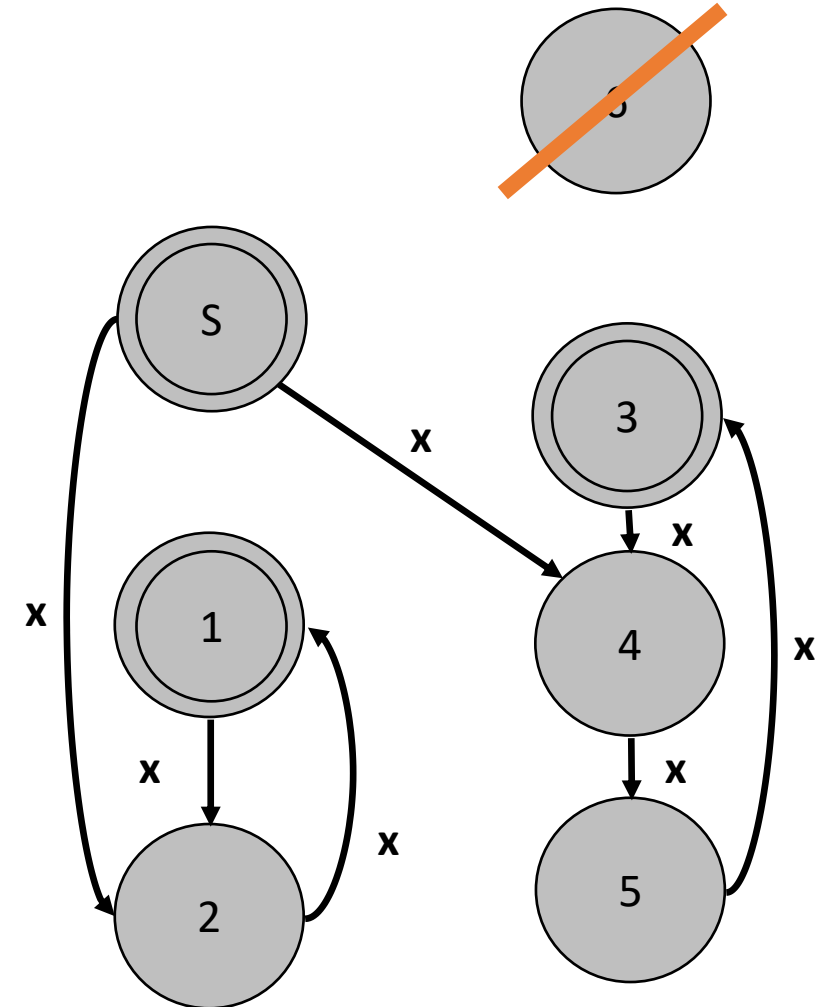
Example, Done!

Eliminating ϵ -Transitions

Can also remove unreachable "useless" state



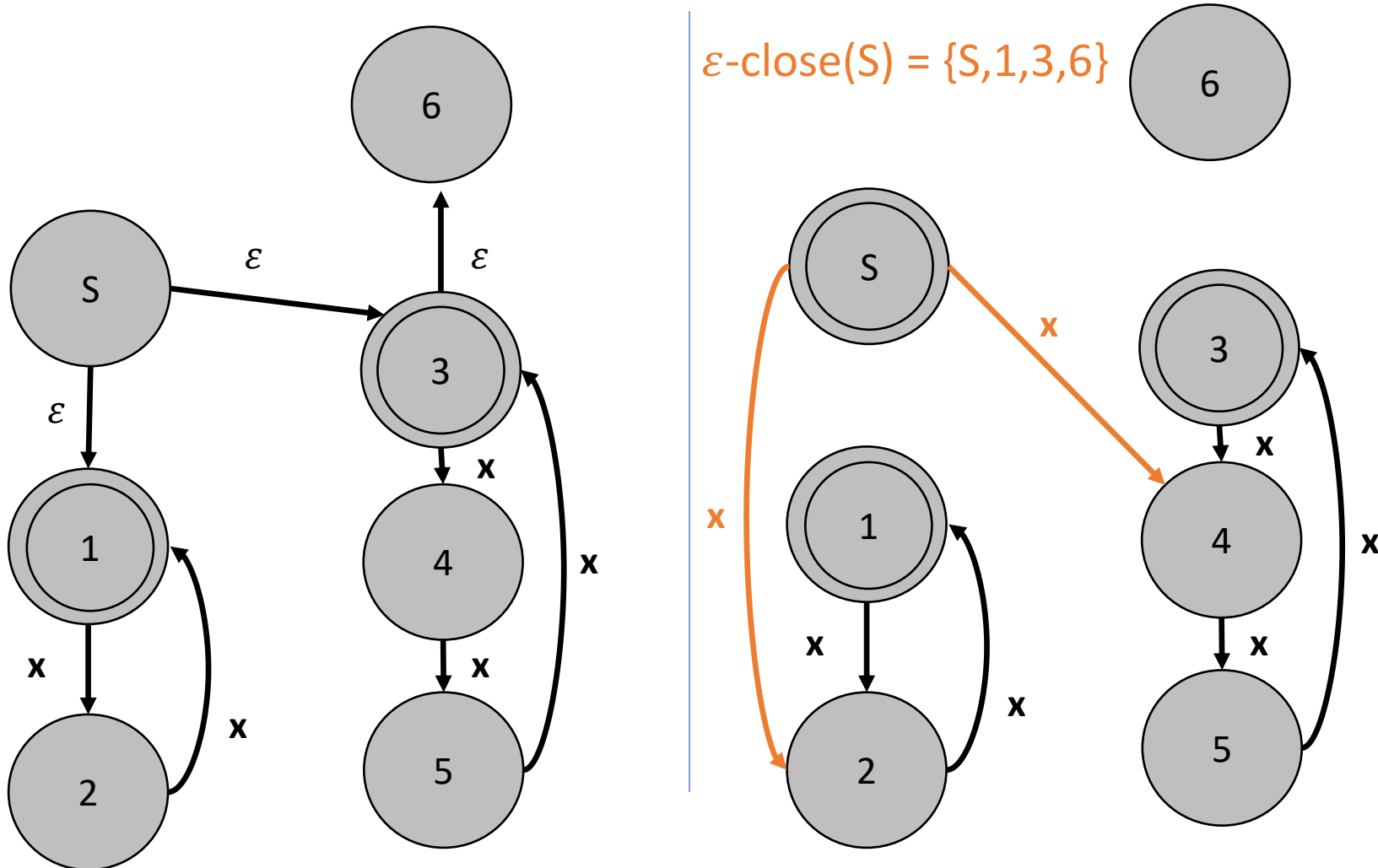
=



Example, Step IV

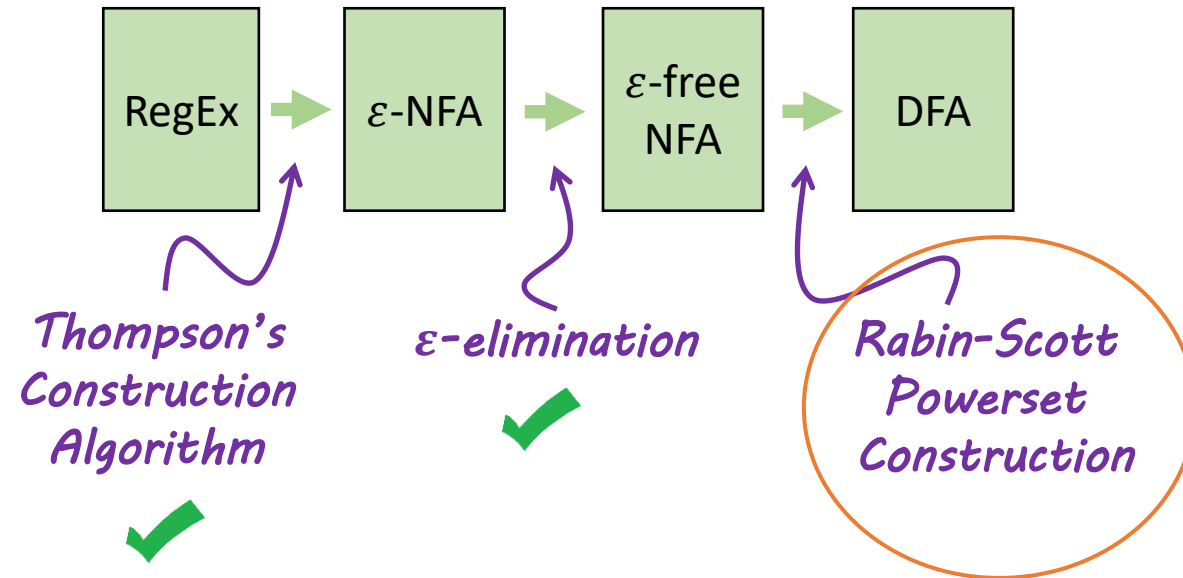
Eliminating ϵ -Transitions

Put $s, c \rightarrow t$ in δ' if there is a c -edge to t in ϵ -close(s)



From RegEx to DFA

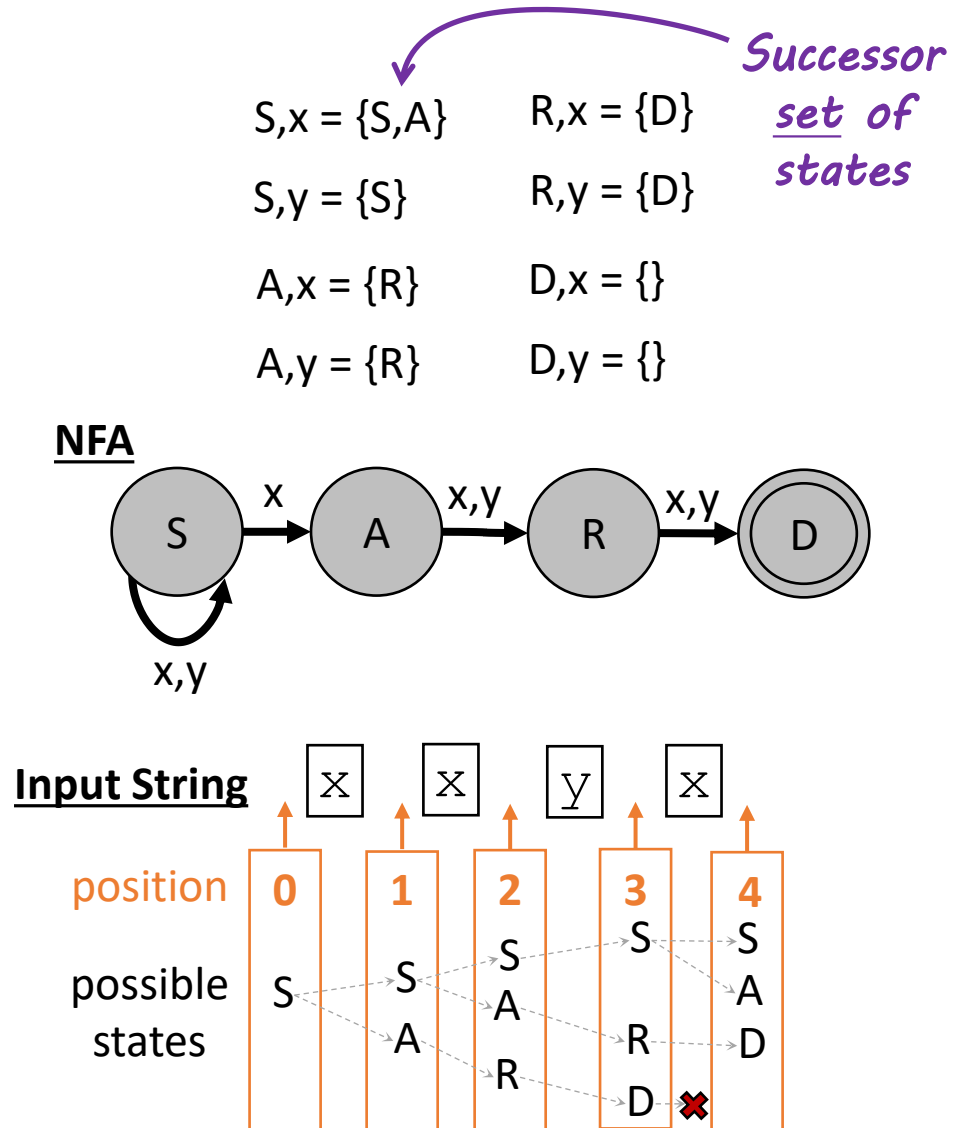
Lecture 2 – Implementing Scanners



Recall: NFA Matching Procedure

Rabin-Scott Powerset construction

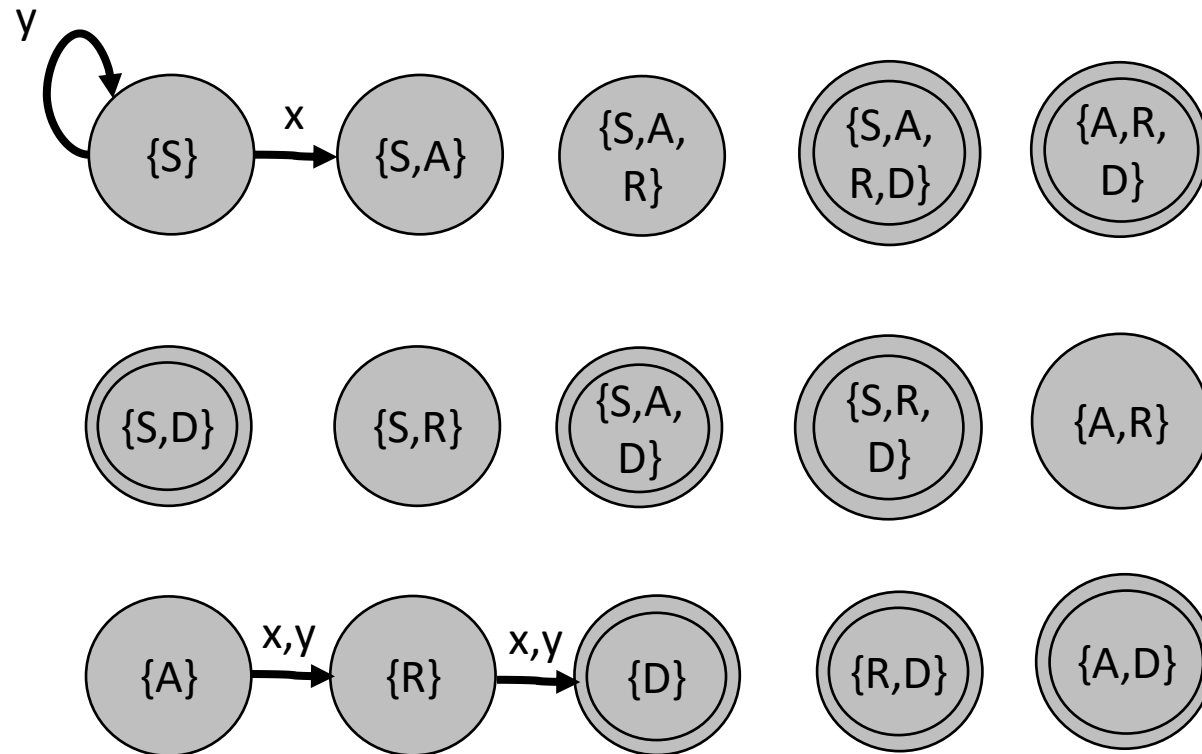
- NFA can “choose” which transition to take
 - Always moves to states that leads to acceptance (if possible)
- Simulate set of states the NFA *could* be in
 - If any state in the ending set is final, string accepted



From Successors to Powerset DFA

Rabin-Scott Powerset Construction

$S, x = \{S, A\}$	$A, x = \{R\}$	$R, x = \{D\}$	$D, x = \{\}$
$S, y = \{S\}$	$A, y = \{R\}$	$R, y = \{D\}$	$D, y = \{\}$



From Successors to Powerset DFA

Rabin-Scott Powerset Construction

$$S, x = \{S, A\}$$

$$A, x = \{R\}$$

$$R, x = \{D\}$$

$$D, x = \{\}$$

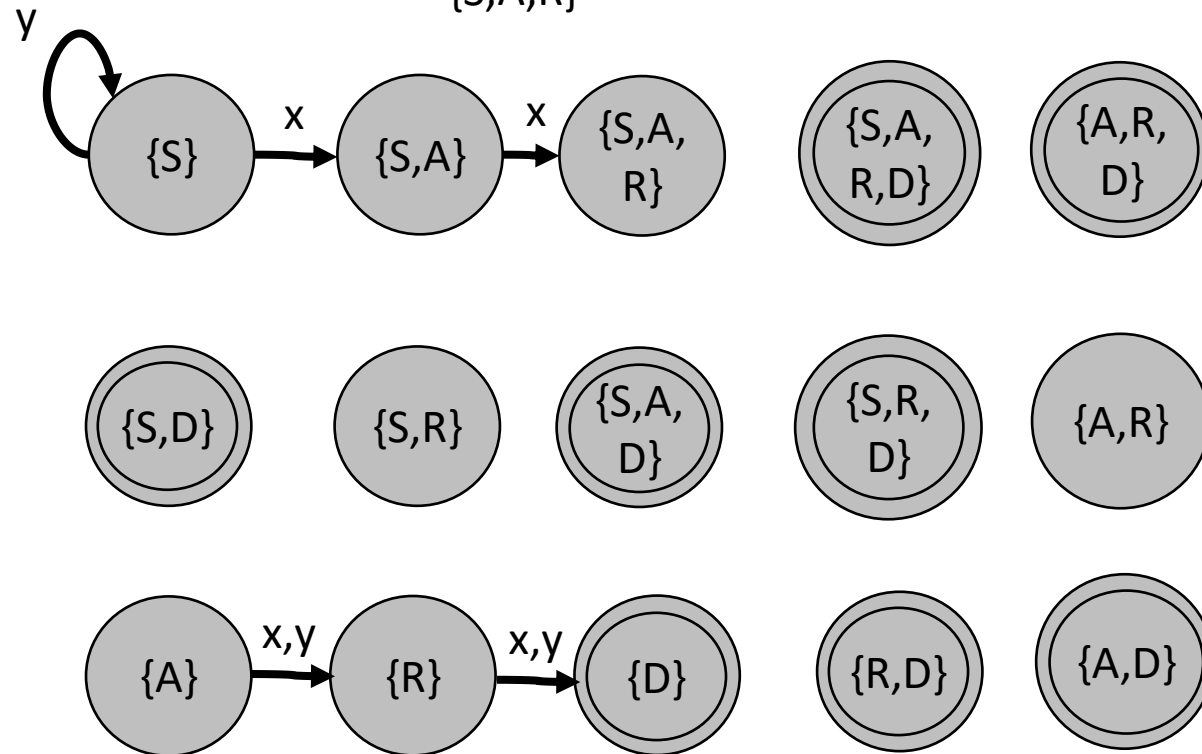
$$S, y = \{S\}$$

$$A, y = \{R\}$$

$$R, y = \{D\}$$

$$D, y = \{\}$$

$$\begin{aligned}\{S, A\}, x &= S, x \cup A, x \\ &= \{S, A\} \cup \{R\} \\ &= \{S, A, R\}\end{aligned}$$

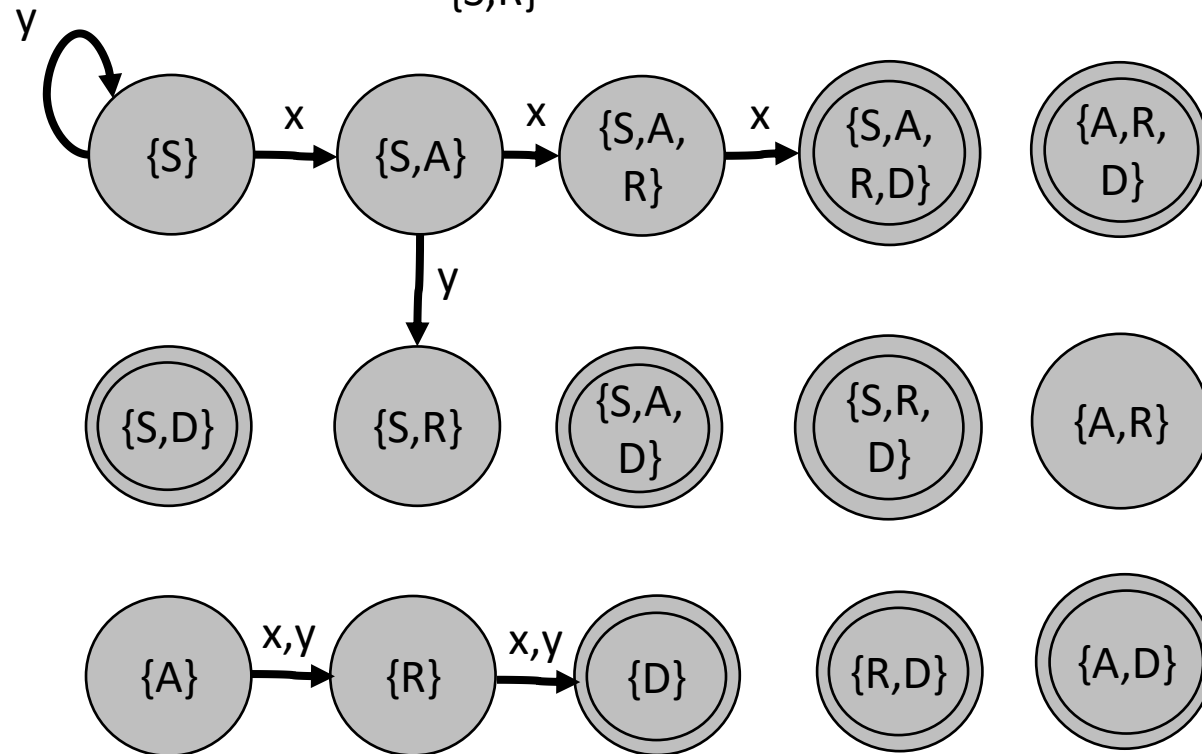


From Successors to Powerset DFA

Rabin-Scott Powerset Construction

$$\begin{array}{llll} S,x = \{S,A\} & A,x = \{R\} & R,x = \{D\} & D,x = \{\} \\ S,y = \{S\} & A,y = \{R\} & R,y = \{D\} & D,y = \{\} \end{array}$$

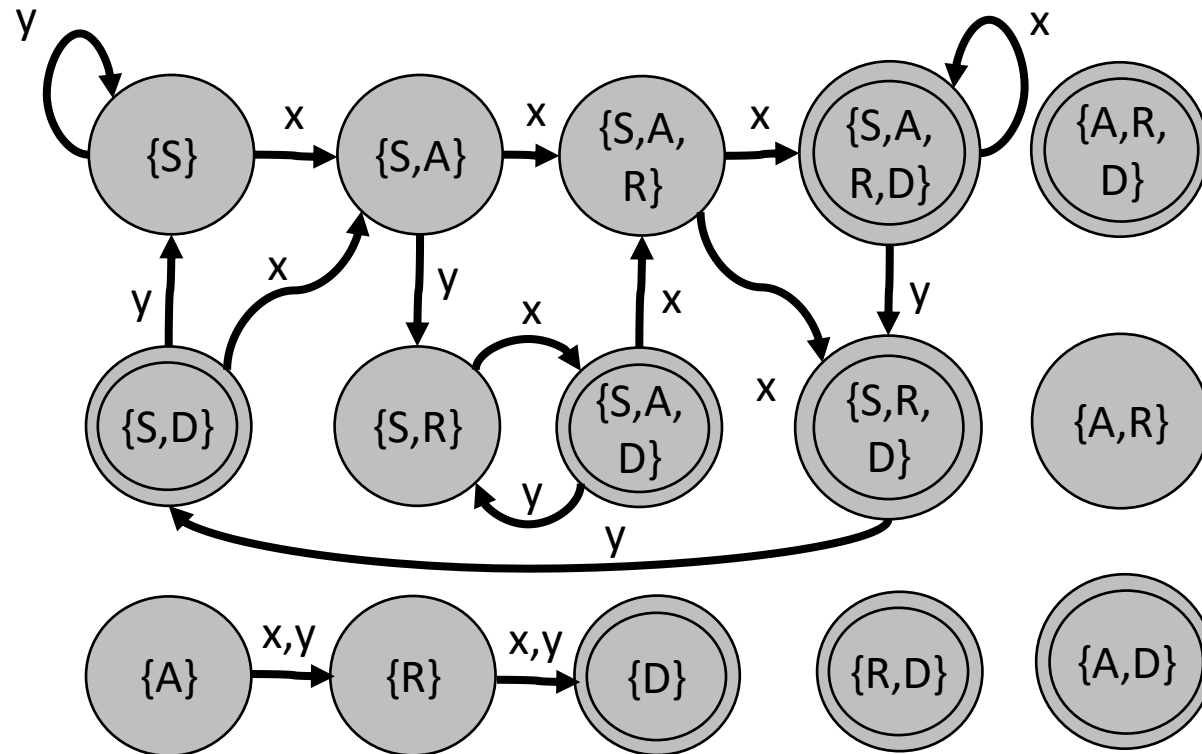
$$\begin{aligned} \{S,A\}, y &= S,y \cup A,y \\ &= \{S\} \cup \{R\} \\ &= \{S,R\} \end{aligned}$$



From Successors to Powerset DFA

Rabin-Scott Powerset Construction

$S, x = \{S, A\}$	$A, x = \{R\}$	$R, x = \{D\}$	$D, x = \{\}$
$S, y = \{S\}$	$A, y = \{R\}$	$R, y = \{D\}$	$D, y = \{\}$

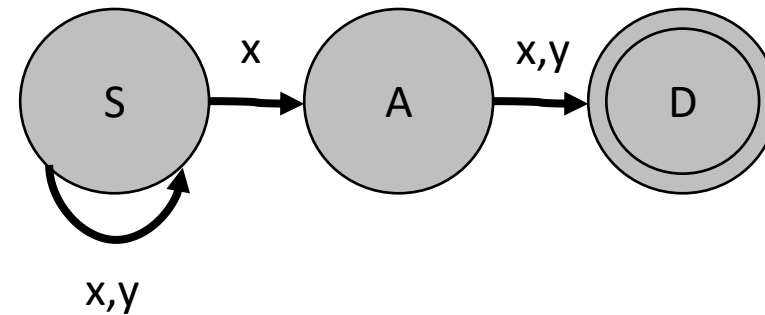


Exponential State Count

Rabin-Scott Powerset Construction

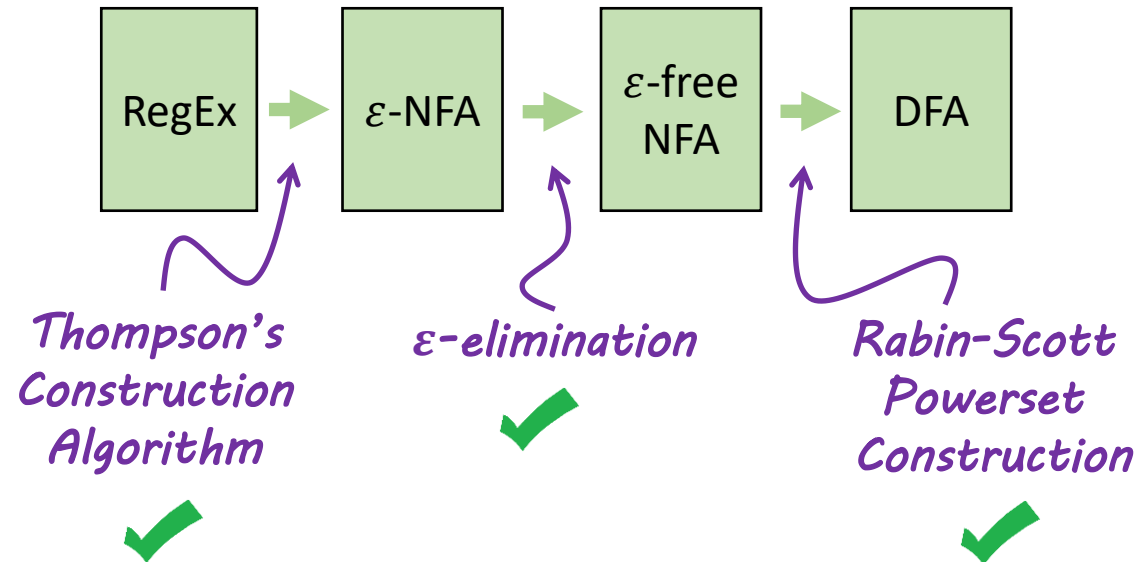
- How many states might the DFA have?
 - $2^{|Q|}$
- Why $2^{|Q|}$?

<u>S</u>	<u>A</u>	<u>D</u>	
0	0	0	{}
0	0	1	{D}
0	1	0	{A}
1	0	0	{S}
0	1	1	{A,D}
1	1	0	{S,A}
1	0	1	{S,D}
1	1	1	{S,A,D}



From RegEx to DFA

Lecture 2 – Implementing Scanners



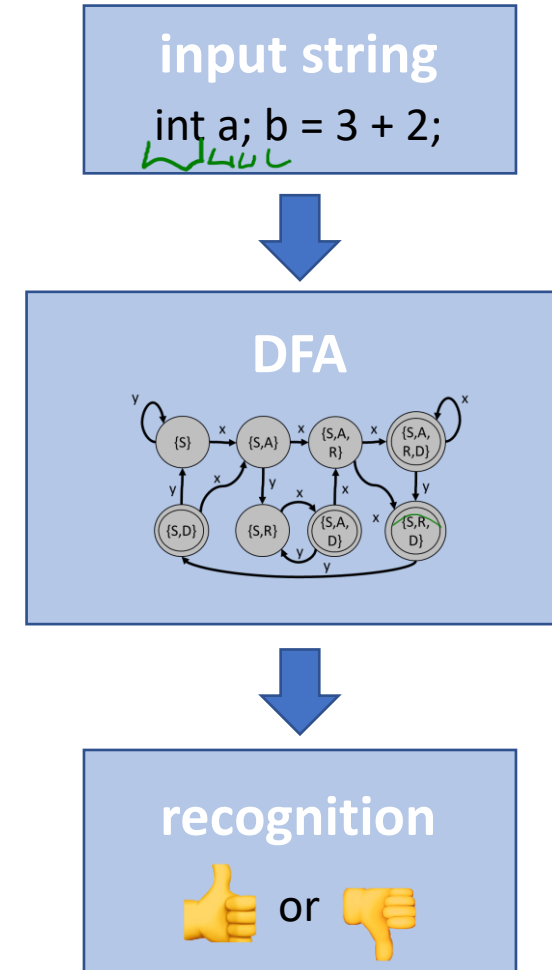
DONE!

... or are we?

DFA $\neq$ Tokenizer

Limitations

- Finite automata only check for language membership of a string (recognition)
- The Scanner needs to
 - Break the input into many different tokens
 - Know what characters comprise the token

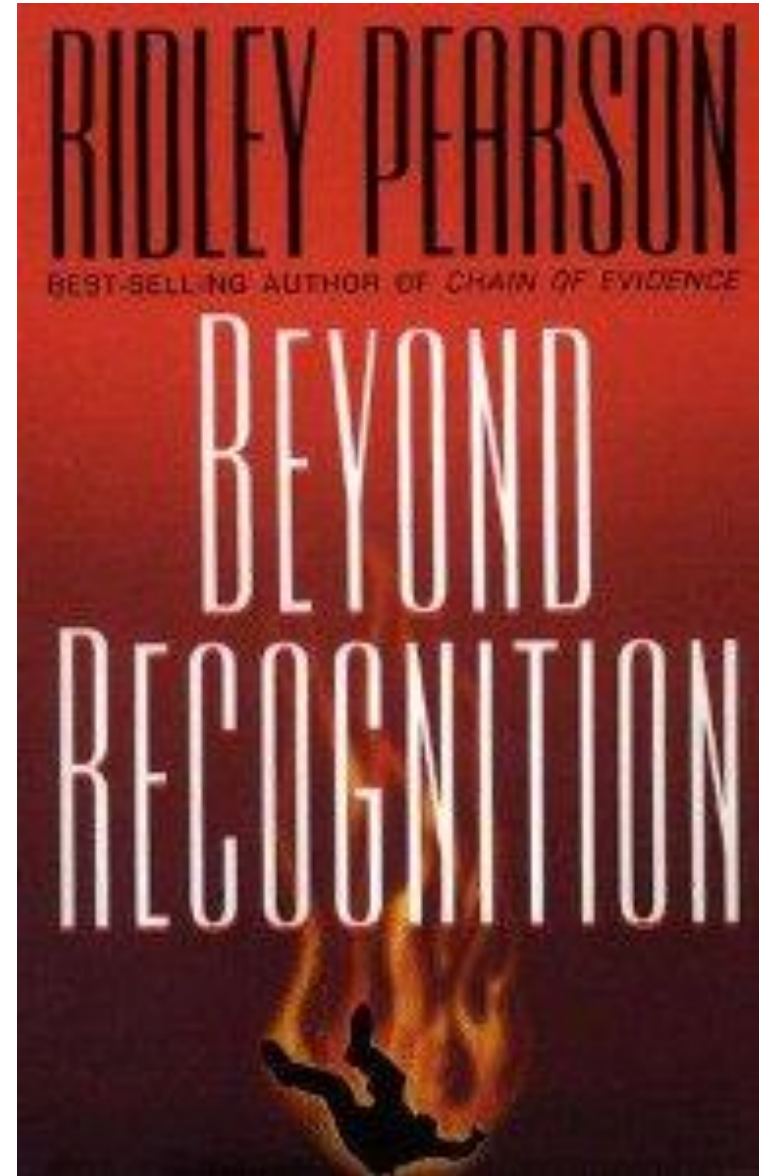


DFA $\neq$ Tokenizer

Limitations

- Finite automata only check for language membership of a string (recognition)
- The Scanner needs to
 - Break the input into many different tokens
 - Know what characters comprise the token

We need to go... *beyond recognition*



Next Time

Lecture 3 Preview

