LEX4: Regexps as Automata

Lexical Analysis

CMPT 379: Compilers
Instructor: Anoop Sarkar
anoopsarkar.github.io/compilers-class

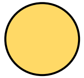
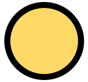
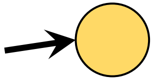
Regular Expressions

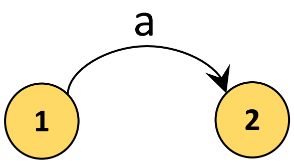
- We write down a **pattern** as a regular expression in order to describe all lexemes for a token
- We need a decision procedure (an algorithm) for matching lexemes
- Given a pattern described as a regexp r and input string s
  - return True if $s \in L(r)$
  - return False if $s \notin L(r)$
- This decision procedure is called a **recognition** algorithm

Regular Expressions

- We will do this by compiling the regular expression into a data structure called a finite state automata (FA)
- Finite state automata can be:
  - Deterministic (DFA)
  - Non-deterministic (NFA)
- DFA and NFA each have their own **recognition** algorithm for matching against an input string.

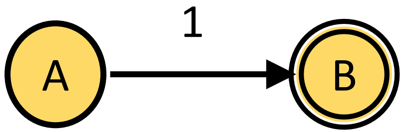
Finite State Automata

- An alphabet Σ of input symbols
- A finite set of states S 
  - One start state q_0 
  - zero or more final (accepting) states F 
- A transition function :
  - $\delta\colon\ S \times \Sigma \Rightarrow S$ 
- Example: δ(1, a) = 2



FA: Example 1

A finite automaton that accepts only ‘1’

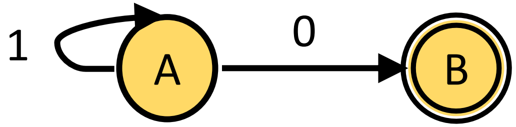


state	input	
A	1	
B	1	Accept
A	0	Reject
A	1 0	
B	1 0	Reject

Language of a FA: set of accepted strings

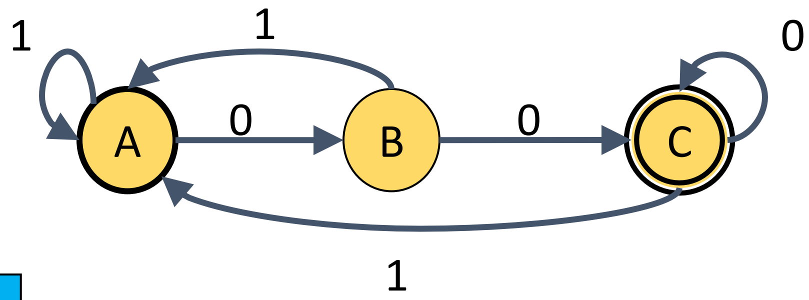
FA: Example 2

A finite automaton accepting any number of 1's followed by a single 0



FA: Example 3

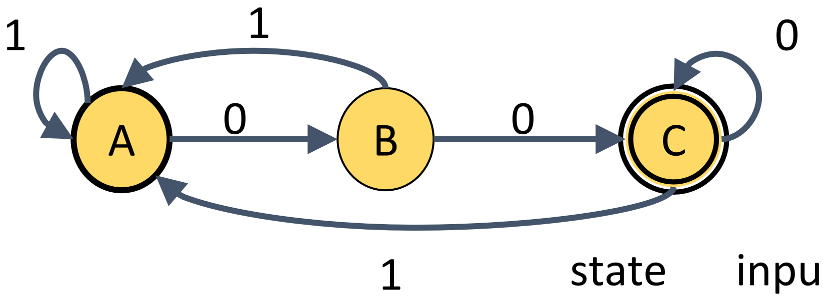
What regular expression does this automaton accept?



A: start state
C: final state

Answer: (0|1)*00

FA simulation == recognition algorithm

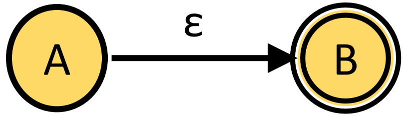


Input string: 00100

state	input	
A	00100	
B	00100	
C	00100	
A	00100	
B	00100	
C	00100	Accept

ε-move

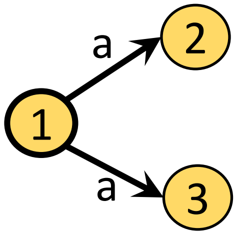
Another kind of transition: ε-moves



state	input
A	$x_1\ x_2\ x_3$
B	$x_1\ x_2\ x_3$

Deterministic Finite Automata (DFA)

Rule 1: One transition per input per state

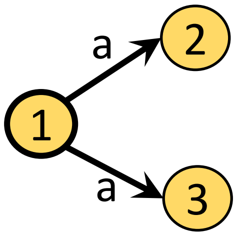


Invalid

Rule 2: No ε-moves

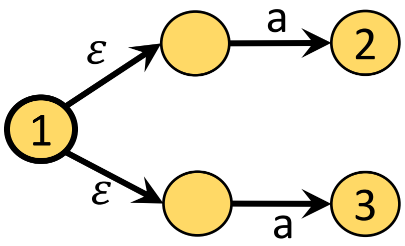
Nondeterministic Finite State Automata (NFA)

Can have multiple transitions for same symbol from a state



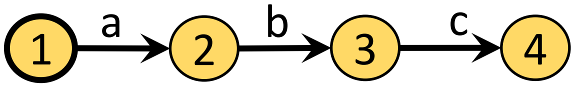
Allowed!

Can have ε-moves

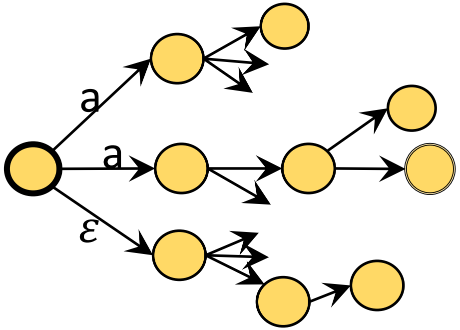


Nondeterministic Finite State Automata (NFA)

A DFA takes only one path through the state graph (per input)



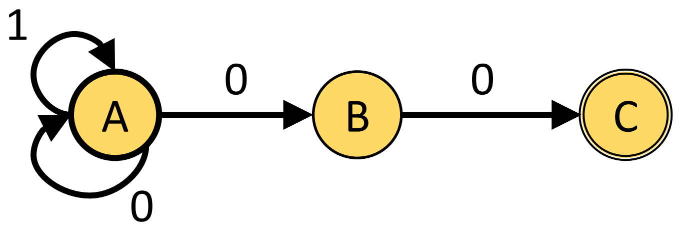
NFA can choose the path!



An NFA accepts the input if any one of the paths leads to a final state.

Nondeterministic Finite State Automata (NFA)

An NFA can reach multiple states simultaneously



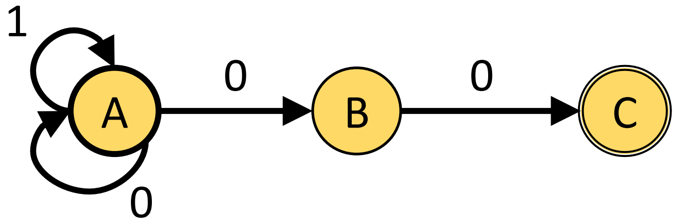
Q: Write down the regexp for this NFA.

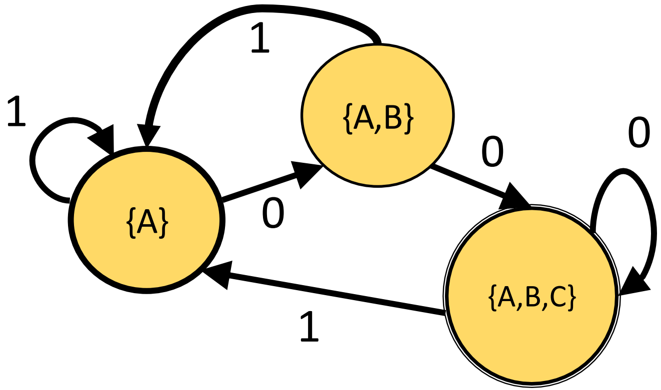
state	input
{A}	100
{A}	100
{A,B}	100
{A,B,C}	100

Q: Draw an NFA for regexp `(0|1)(0|1)*`

Nondeterministic to Deterministic

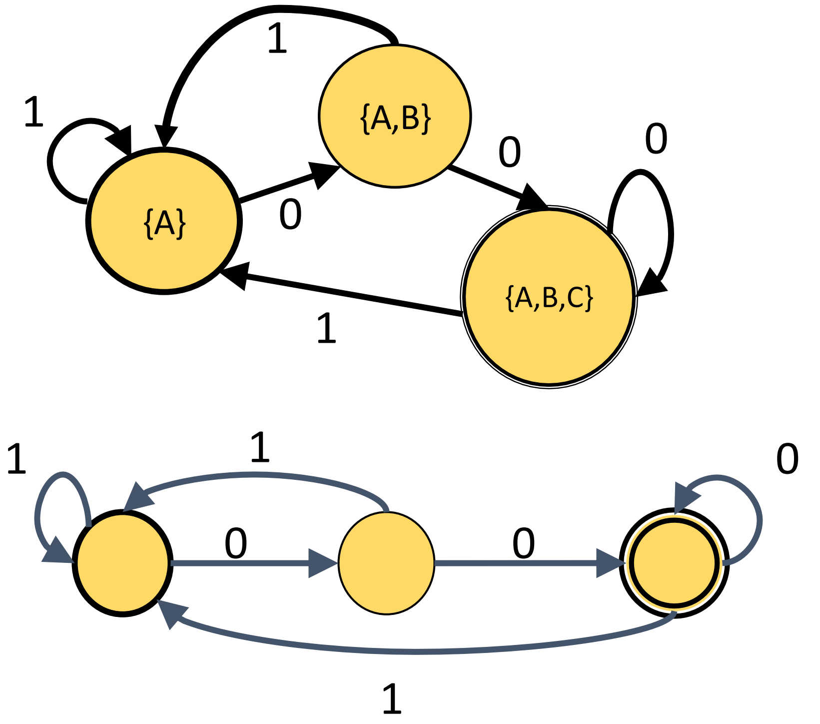
The Subset Construction converts an NFA into a DFA

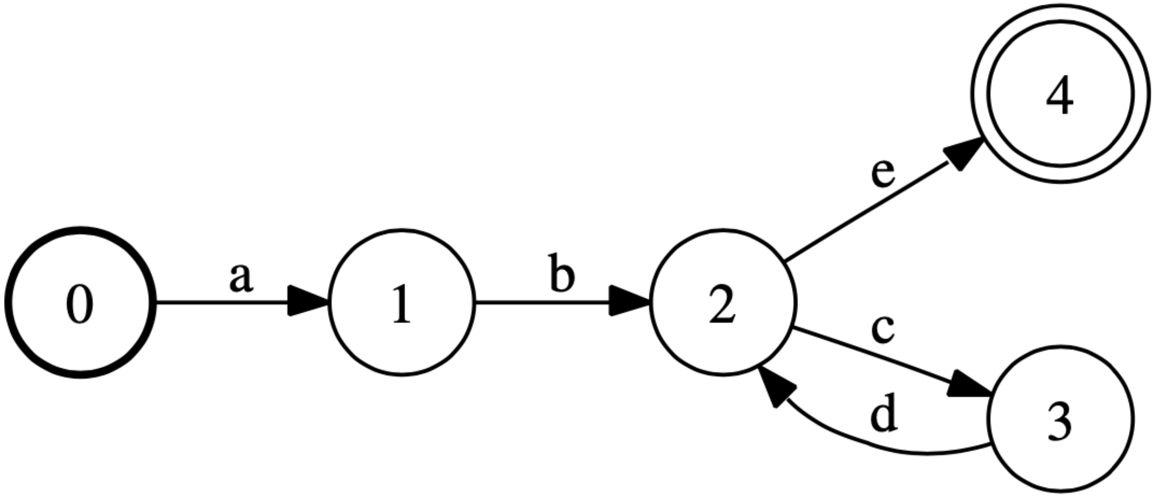




DFA state, Σ	DFA state
{A}, 0	{A, B}
{A}, 1	{A}
{A, B}, 0	{A, B, C}
{A, B}, 1	{A}
{A, B, C}, 0	{A, B, C}
{A, B, C}, 1	{A}

Nondeterministic to Deterministic





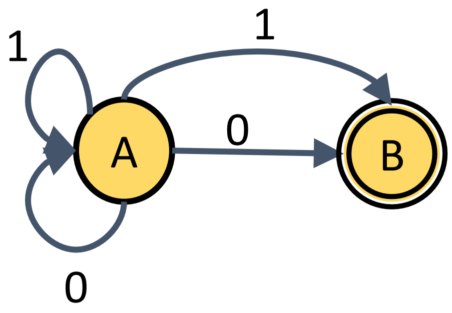
Q: Write down the regexp for this DFA.

NFAs vs DFAs

- NFAs and DFAs recognize the same set of languages
  - Regular languages, the languages `L(r)` where `r` is a regular expression
- DFAs are faster to execute
  - There are no choices to consider – it is deterministic (hence the name)
- DFAs usually have fewer states than NFAs
- But in a worst-case analysis, DFAs can be larger than NFAs
  - Exponentially larger

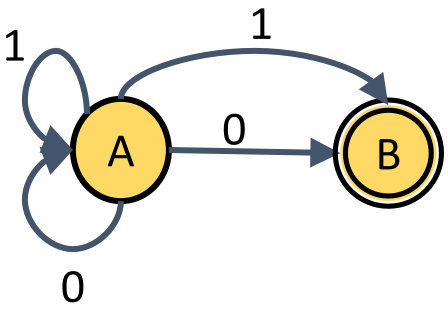
Extra Slides

Nondeterministic Finite State Automata (NFA)

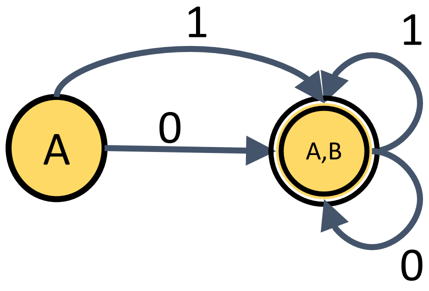


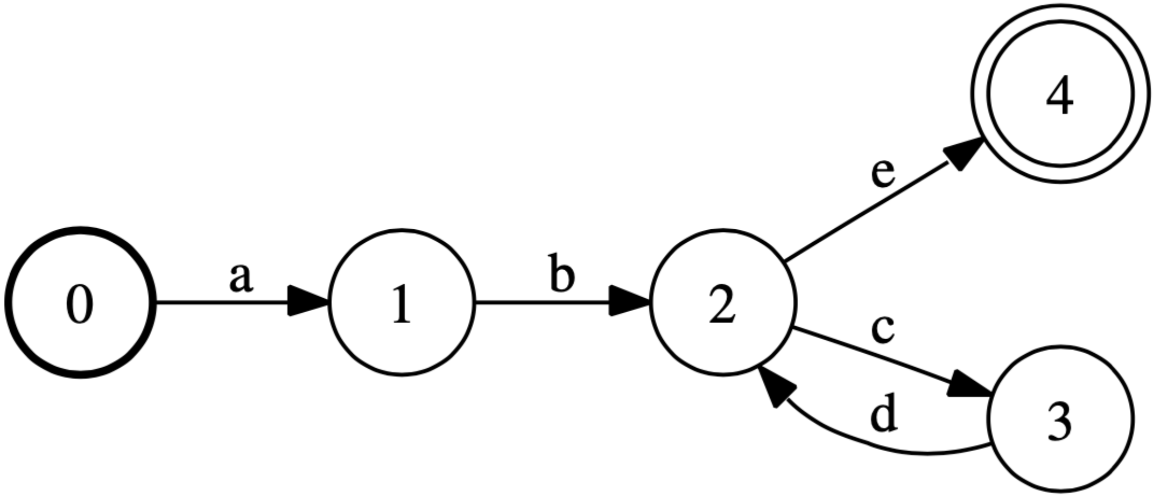
Q: Draw an NFA for regexp `(0|1)(0|1)*`

Nondeterministic to Deterministic



DFA state, Σ	DFA state
{A}, 0	{A, B}
{A}, 1	{A, B}
{A, B}, 0	{A, B}
{A, B}, 1	{A, B}





Q: Write down the regexp for this DFA.

A: `ab(cd)*e`