CS 121: Lecture 8
Finite Automata and Regular Functions

Adam Hesterberg

https://madhu.seas.Harvard.edu/courses/Fall2020

Book: https://introtcs.org

How to contact us:
- The whole staff (faster response): CS 121 Piazza
- Only the course heads (slower): cs121.fall2020.course.heads@gmail.com

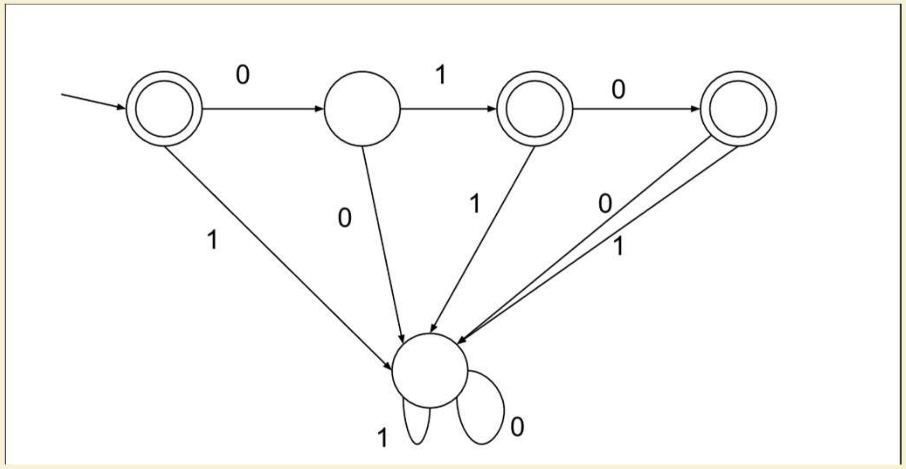
Today

- Comparison of regular expressions and finite automata
- Nondeterministic Finite Automata
- Preview of next lecture: non-regular functions

Reminder: Regular Expressions

- Defines function $f: \{0,1\}^* \rightarrow \{0,1\}$
- Definition:
  - Basic cases:
    - 0, 1, $\phi = \{\}$ (empty set), "" $= \varepsilon$ (null string)
  - Compound cases: If r_1, r_2 are regular expressions, then so are:
    - $r_1 r_2$: "r_1 followed by r_2" (or "concatenation")
    - $(r_1 | r_2)$: "$r_1$or r_2"
    - r_1^*: "Concatenation of nonnegative (finite) number of r_1's"
- Example:
  - 0|(1(0|1)*0): nonnegative even integers in binary
  - (deterministic |)finite(-state | state |)automaton

Reminder: Deterministic Finite Automata (DFAs)



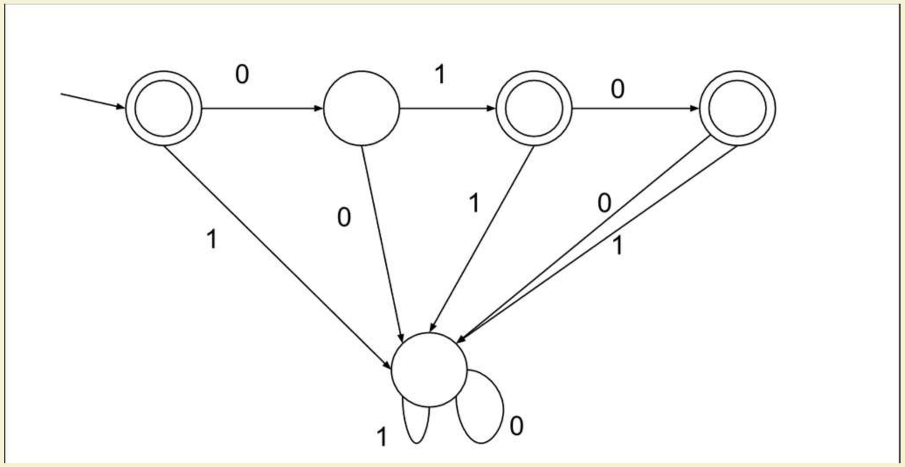
- Computes function $f: \{0,1\}^* \to \{0,1\}$
- Specification:
  - accept states S (subset of all states, C)
  - transition function $\mathrm{C} \times \{0,1\} \to C$
- Operation:
  1. Starts in state 0
  2. Read one bit of input x_0: do the state transition matching current state and just-read input.
  3. Move past just-read input.
  4. If input not done, repeat from Step 2.
  5. When done: Accept (output 1) if the sequence of transitions ends in an accept state $q \in S$ and reject (output 0) otherwise.

Comparison: DFAs and Regular expressions

	Regular Expressions	DFAs
Define function $f\colon \{0,1\}^* \to \{0,1\}$		
Define set of strings $S \subseteq \{0,1\}^*$		
One for every finite set		
If S_1 is computed by one, so is S_1^*		
If f_1 is comp by one, so is $\mathrm{NOT}(f_1)$		
If f_1 and f_2 are, so is $\mathrm{OR}(f_1, f_2)$		
If f_1 and f_2 are, so is $\mathrm{AND}(f_1, f_2)$		
If S_1 and S_2 are, so is $S_1 S_2 = \{s\colon \exists s_1 \in S_1. \exists s_2 \in S_2. s = s_1 s_2\}$		

If f_1 is the function of some DFA, so is NOT(f_1)

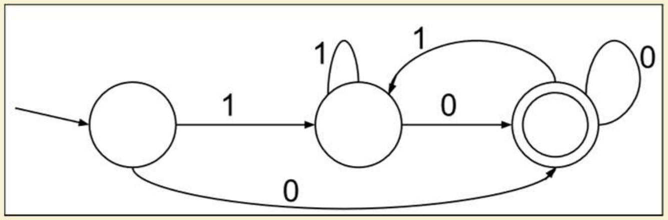
DFA for the function that's 1 at "", "01", "010", and nothing else:



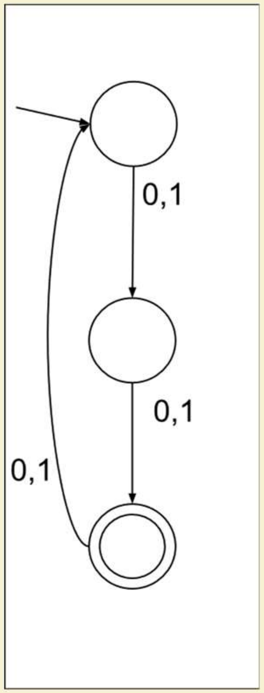
DFA for the function that's 1 at everything but "", "01", and "010"?

If f_1 and f_2 are DFA functions, is AND(f_1, f_2)?

DFA for multiples of 2 in binary:



DFA for strings of length 2 mod 3:



Is there a DFA for multiples of 2 of length 2 mod 3?

Exercise Break 1:

1) Express “if each of f_1 and f_2 is the function computed by some DFA, so is $\mathrm{OR}(f_1, f_2)$’’ in terms of sets of strings instead of functions.

2) Prove the above.

3) Prove that if each of f_1 and f_2 is the function computed by some DFA, so is $\mathrm{NAND}(f_1, f_2)$.

4) True or false: 3) means that every function is the function computed by some DFA.

DFA for each infinite function?

True or false: "if each of f_1 and f_2 is the function computed by some DFA, so is $\mathrm{NAND}(f_1, f_2)$" means that every infinite function is the function computed by some DFA.

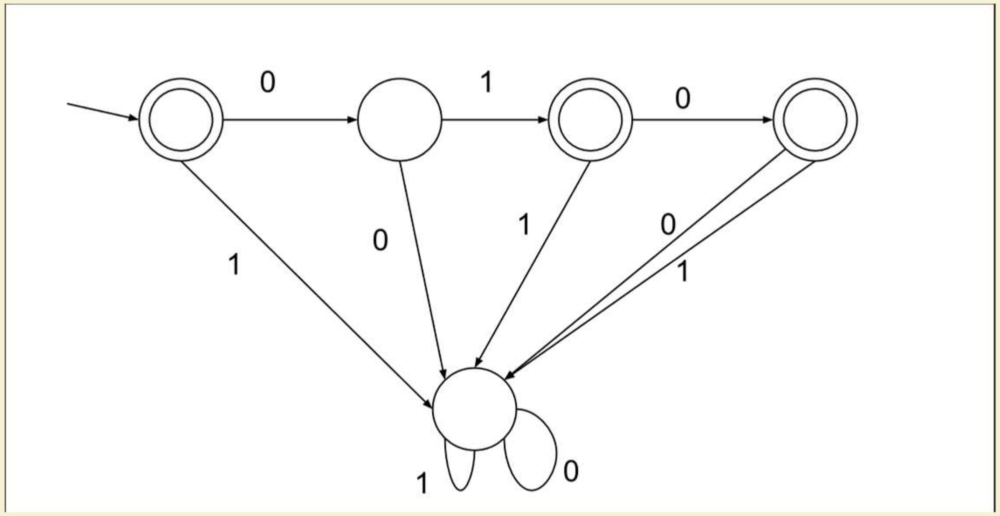
- If f_1 and f_2 are the function of DFAs with q_1 and q_2 states, $\mathrm{NAND}(f_1, f_2)$ is the function of some DFA with $q_1 q_2$ states.
- NAND of finitely many functions: still function of some DFA.
- NAND of infinitely many functions: DFAs aren't allowed infinitely many states!

Comparison: DFAs and Regular expressions

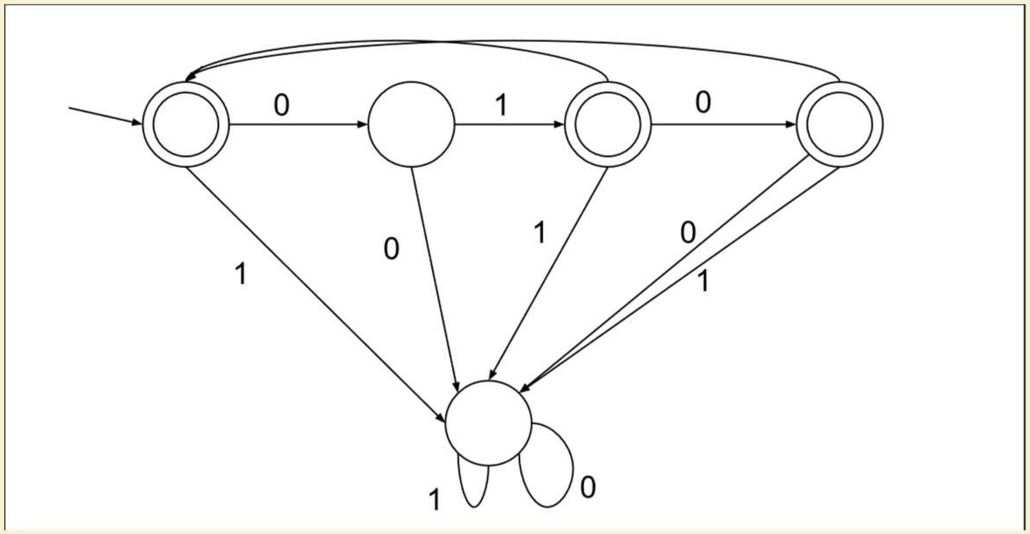
	Regular Expressions	DFAs
Define function $f: \{0,1\}^* \rightarrow \{0,1\}$	Yes	Yes
Define set of strings $S \subseteq \{0,1\}^*$	Yes	Yes
One for every finite set	Yes	Yes
If S_1 is computed by one, so is S_1^*	Yes	
If f_1 is comp by one, so is $\mathrm{NOT}(f_1)$		Yes
If f_1 and f_2 are, so is $\mathrm{OR}(f_1, f_2)$	Yes	Yes
If f_1 and f_2 are, so is $\mathrm{AND}(f_1, f_2)$		Yes
If S_1 and S_2 are, so is $S_1 S_2 = \{s: \exists s_1 \in S_1 . \exists s_2 \in S_2 . s = s_1 s_2\}$	Yes	

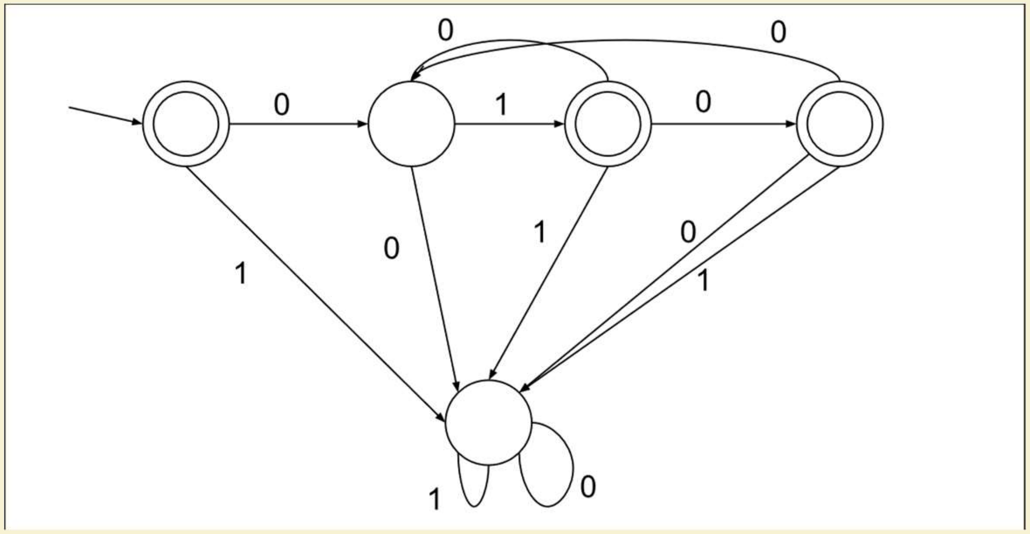
Kleene closure for DFAs?

At right is a DFA that accepts (|01|010):



Which of the bottom two is a DFA that accepts (|01|010)*?

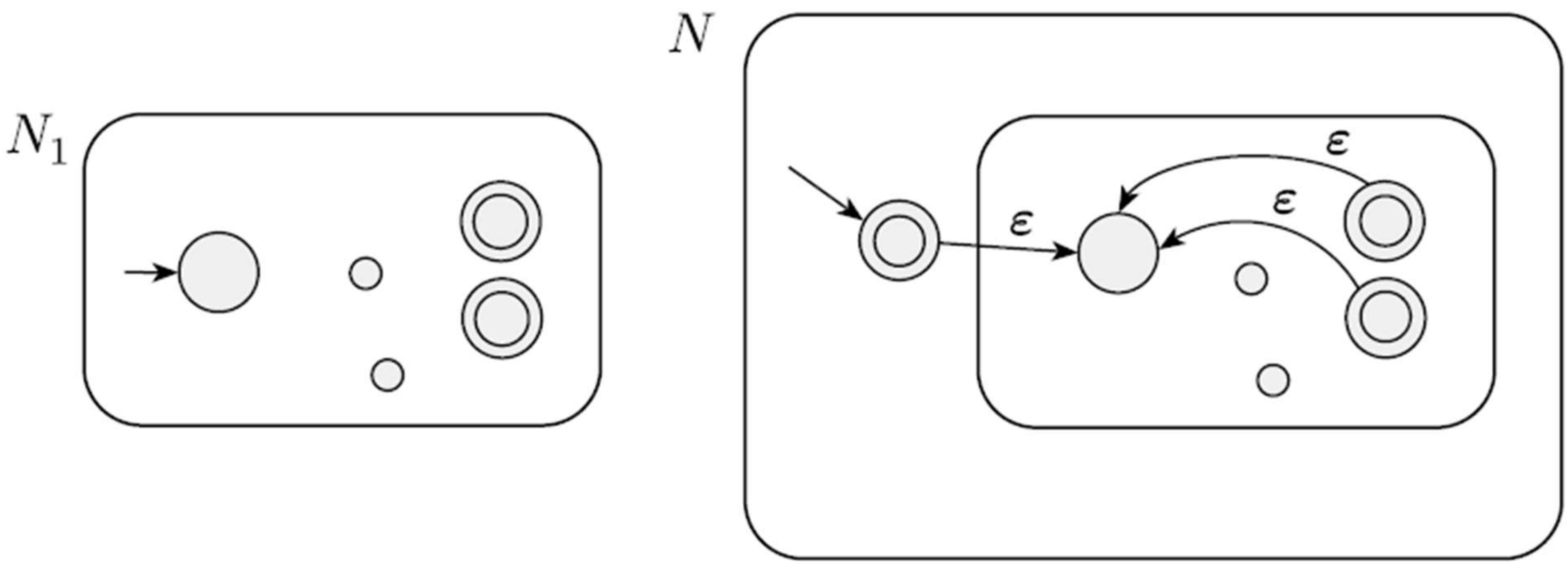




Non-deterministic Finite Automata (NFAs)

- Defines ~~Computes~~ function $f: \{0,1\}^* \to \{0,1\}$
- Specification:
  - accept states S (subset of all states, C)
  - transition ~~function~~ relation $\mathrm{C} \times \{0,1, \varepsilon = ""\} \to C$
- Operation:
  1. Starts in state 0
  2. Read up to one bit of input x_0: do ~~the~~ **any** state transition matching current state and just-read input.
  3. Move past just-read input.
  4. If input not done, repeat from Step 2.
  5. When done: Accept (output 1) if ~~the~~ **any** sequence of transitions ends in an accept state $q \in S$ and reject (output 0) otherwise.

Kleene closure for NFAs?

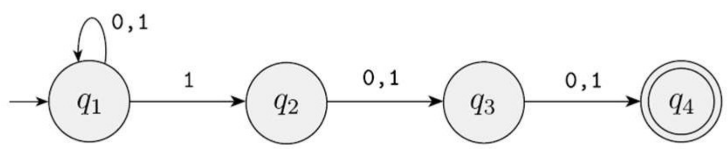


DFA-NFA equivalence

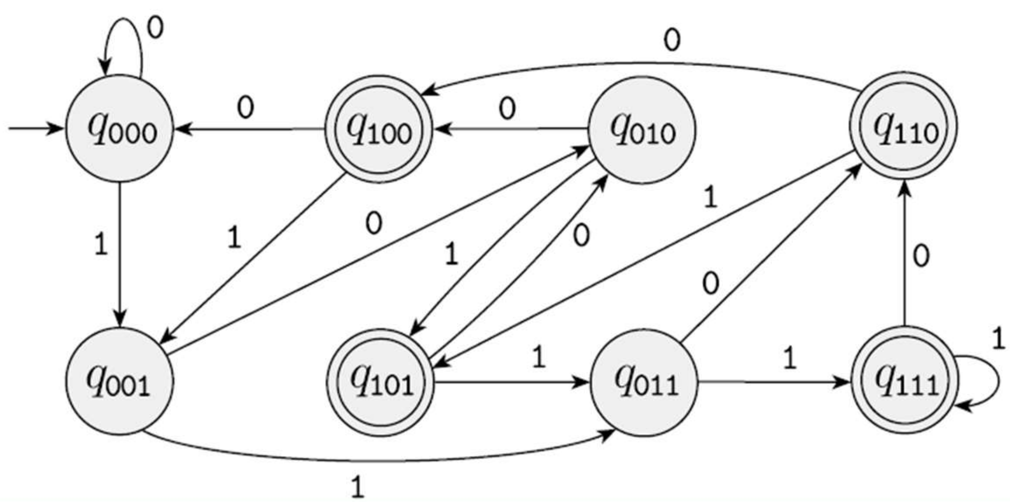
Theorem: For every NFA, there's a DFA that accepts the same language.

Proof: As an NFA reads its input, at all times, there's a subset of states it could be in. Make each subset of NFA states a DFA state; define DFA transitions and accept states accordingly.

Example NFA (third-last bit 1):



Equivalent DFA:



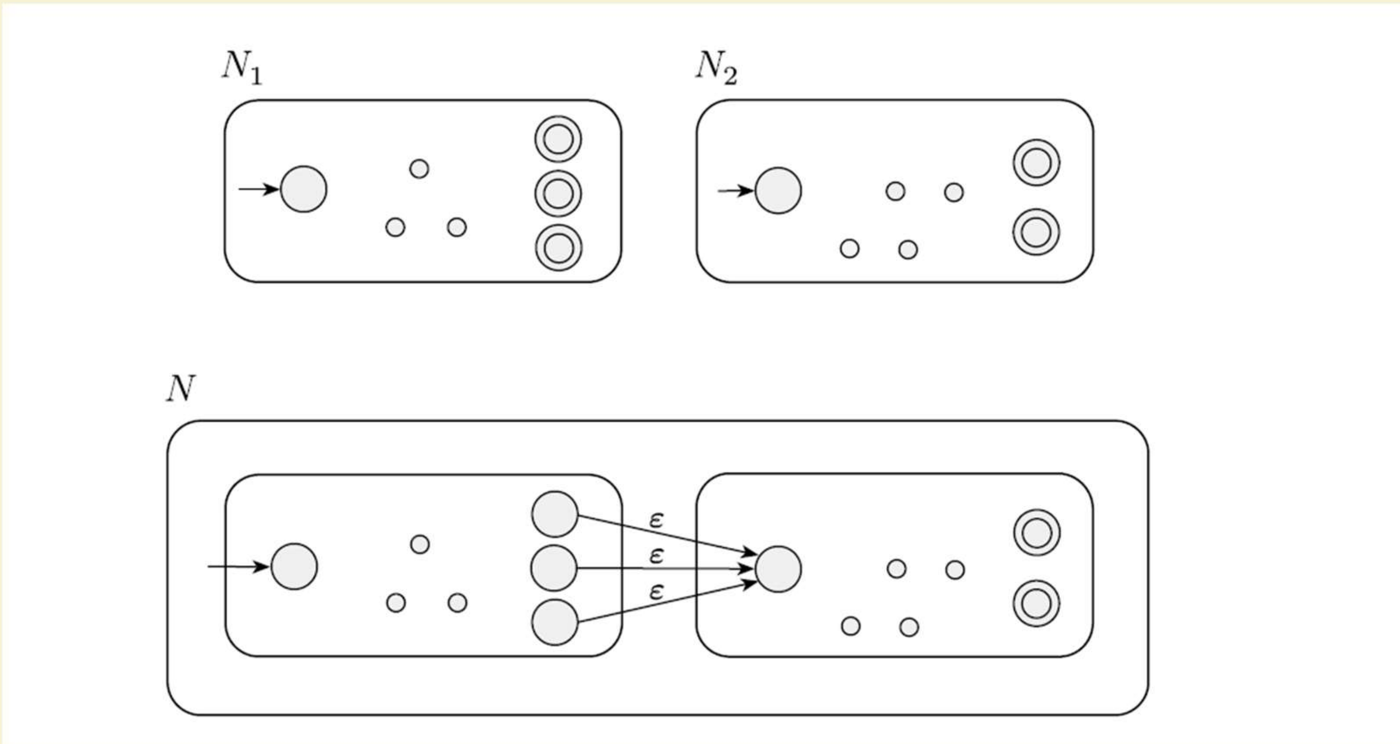
Kleene closure for DFAs, take 2

Exercise Break 2:

1) We negated the function computed by a DFA by switching accept and reject states. Switching accept and reject states doesn't necessarily negate the function defined by an NFA. Why not?

2) If S_1 and S_2 are sets accepted by DFAs D_1 and D_2, prove that $S_1 S_2$ (the set of concatenations of a string in S_1 and a string in S_2) is the set accepted by some DFA. (Hint: Convert to NFAs, solve the same problem for them, and convert back.)

NFA concatenation



Comparison: DFAs and Regular expressions

	Regular Expressions	DFAs
Define function $f\colon \{0,1\}^* \to \{0,1\}$	Yes	Yes
Define set of strings $S \subseteq \{0,1\}^*$	Yes	Yes
One for every finite set	Yes	Yes
If S_1 is computed by one, so is S_1^*	Yes	Yes
If f_1 is comp by one, so is $\mathrm{NOT}(f_1)$		Yes
If f_1 and f_2 are, so is $\mathrm{OR}(f_1, f_2)$	Yes	Yes
If f_1 and f_2 are, so is $\mathrm{AND}(f_1, f_2)$		Yes
If S_1 and S_2 are, so is $S_1S_2 = \{s\colon \exists s_1 \in S_1. \exists s_2 \in S_2. s = s_1s_2\}$	Yes	Yes

Summary: regular expressions vs DFAs

Theorem: For every regular expression, there's an equivalent DFA.

Proof: Regular expressions are built up with *, |, concatenation. Do those with DFAs (possibly via NFAs), as on previous slides.

Theorem: For every DFA, there's an equivalent regular expression, too!

Proof (optional, skipped slides):

- Generalize DFAs/NFAs to allow transitions to be any regular expressions.
- For any DFA/NFA/generalized NFA, eliminate states one by one.
- If just 1 start state and 1 accept state, can read off a regular expression.

Equivalent: DFAs and Regular expressions

	Regular Expressions	DFAs
Define function $f: \{0,1\}^* \to \{0,1\}$	Yes	Yes
Define set of strings $S \subseteq \{0,1\}^*$	Yes	Yes
One for every finite set	Yes	Yes
If S_1 is computed by one, so is S_1^*	Yes	Yes
If f_1 is comp by one, so is $\mathrm{NOT}(f_1)$	Yes	Yes
If f_1 and f_2 are, so is $\mathrm{OR}(f_1, f_2)$	Yes	Yes
If f_1 and f_2 are, so is $\mathrm{AND}(f_1, f_2)$	Yes	Yes
If S_1 and S_2 are, so is $S_1 S_2 = \{s: \exists s_1 \in S_1. \exists s_2 \in S_2. s = s_1 s_2\}$	Yes	Yes

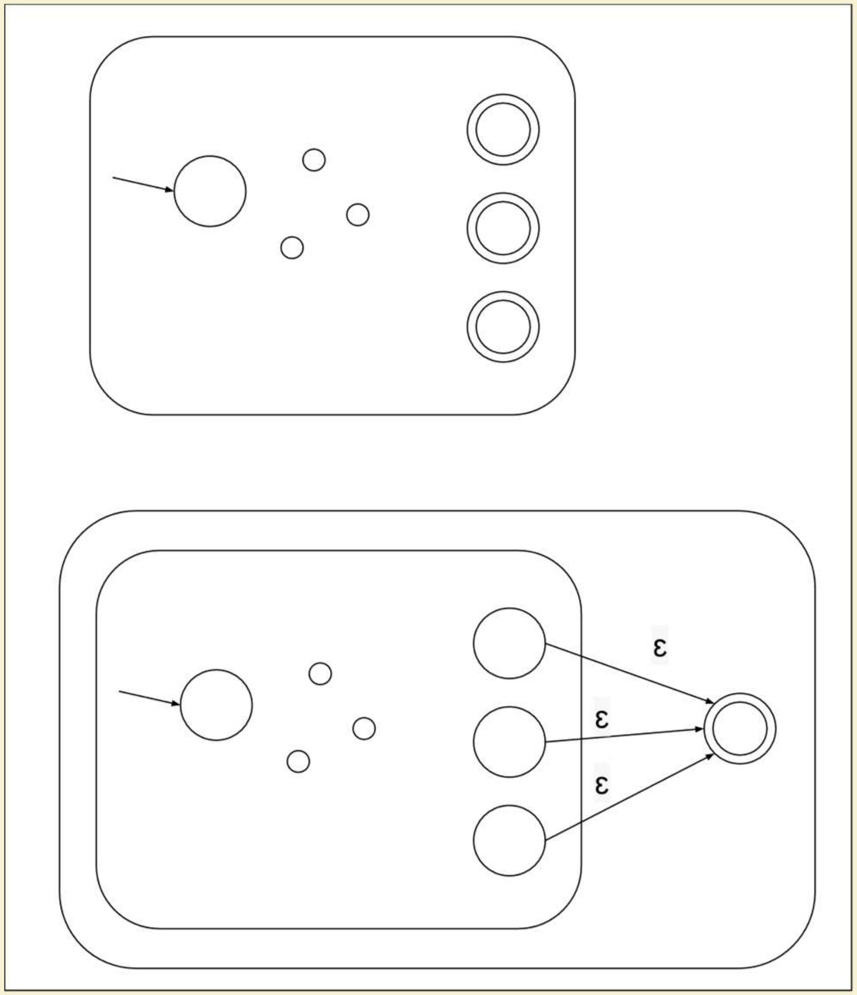
Generalized Non-deterministic Finite Automata

- Defines ~~Computes~~ function $f: \{0,1\}^* \rightarrow \{0,1\}$
- Specification:
  - accept states S (subset of all states, C)
  - transition ~~function~~ relation $\mathrm{C} \times \{0,1, \varepsilon, \mathit{regular\ expressions}\} \rightarrow C$
- Operation:
  1. Starts in state 0
  2. Read up to one or more bits of input: do ~~the~~ **any** state transition matching (as a regular expression) current state and just-read input.
  3. Move past just-read input.
  4. If input not done, repeat from Step 2.
  5. When done: Accept (output 1) if ~~the~~ **any** sequence of transitions ends in an accept state $q \in S$ and reject (output 0) otherwise.

Eliminating all but one accept state of NFAs

Given an NFA with multiple accept states:

- Make a new accept state.
- Add a free transition from each old accept state.
- Un-accept the old accept states.

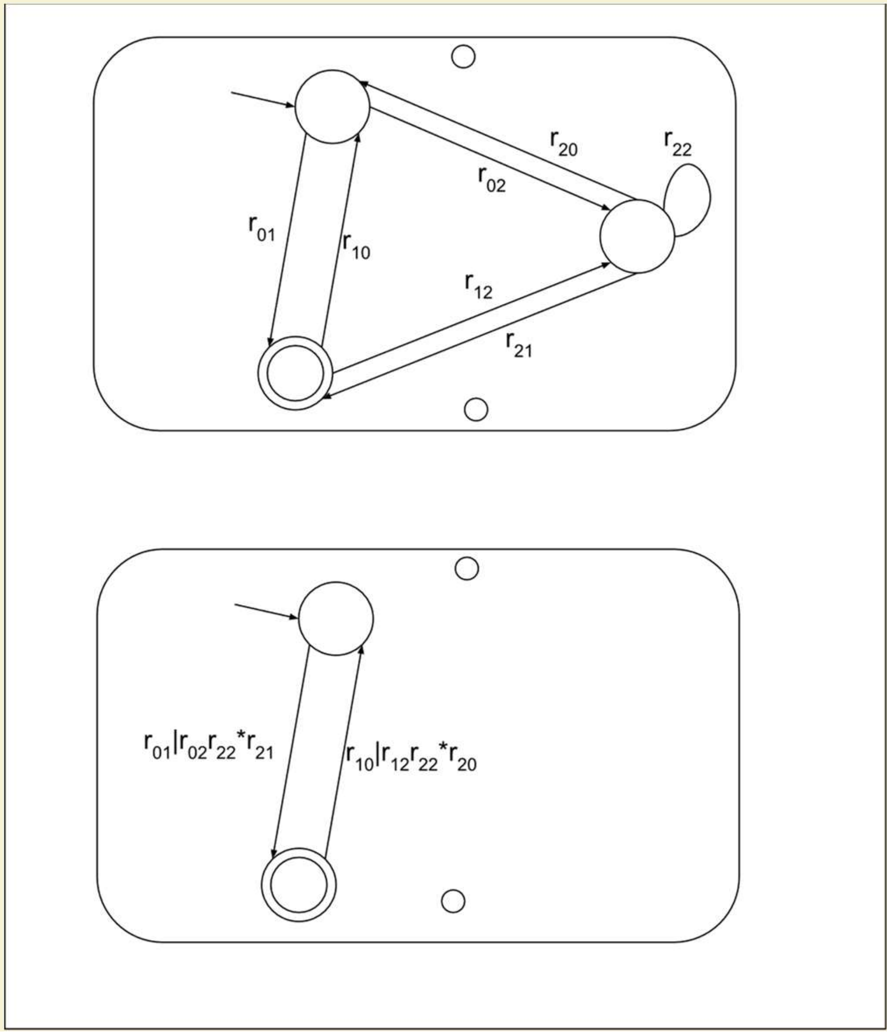


Eliminating non-accept, non-start of gNFAs

Given a gNFA with a non-accept, non-start state c:

- Eliminate it.
- For each ordered pair (a,b) of other states, if:
  - $r_{a,b}$ was the regular expression describing transitions from a to b,
  - $r_{a,c}$, $r_{c,c}$, and $r_{c,a}$ describe transitions from a to c, c to c, and c to a

  then replace $r_{a,b}$ by $r_{a,b}|r_{a,c}r_{c,c}^{*}r_{c,b}$: ways to transition from a to b, possibly through c.



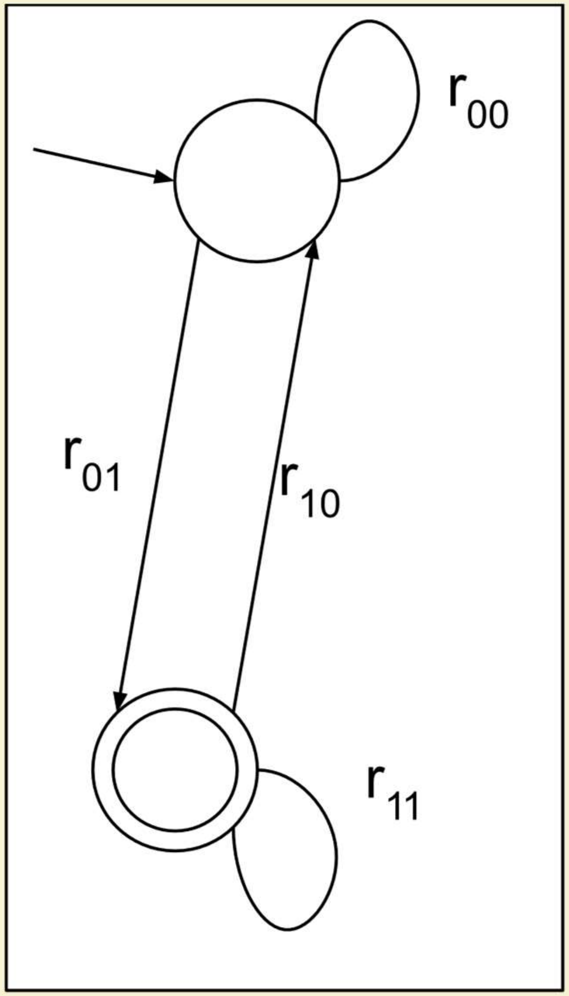
Reading regular expression from 2-state gNFA

Given a gNFA with one start state and one accept state:

A regular expression equivalent to it is:

$$r_{00}^* r_{01} r_{11}^* (r_{10} r_{00}^* r_{01} r_{11}^*)^*$$

So, every NFA accepts the same set of strings
as some regular expression!



Next lecture:

- Recap of DFA-regexp equivalence
- Limits of DFA
  - NAND circuits computed all (finite) functions.
  - Do DFA compute all (infinite) functions? No.
  - What are some functions that are not computed by DFA?