CSCI 361 Lecture 4:
Regular Expressions

Shikha Singh

Announcements & Logistics

- **Assignment 2** due Sept 17 (tomorrow) at 10 pm
  - Please check LaTeX formatting
  - Please assign pages to each question on Gradescope
- Hand in Exercise #3, pick up Exercise #4
- Assignment 1 graded feedback will be released soon
- Reminder: Midterm 1 (Oct 7) and Midterm 2 (Nov 6)
- **Questions?**

Last Time

- Defined non-deterministic finite automata
  - Relaxed transition rules compared to DFAs
  - Equivalent in power
- Practice with NFAs
- Showed regular languages are closed under concatenation using NFAs

Today

- More closure: Kleene star operation
- Regular expressions and equivalence with DFAs/NFAs

Kleene Star

- Let A be a language on Σ
- Definition. Kleene star of A, denoted A^* is defined as:

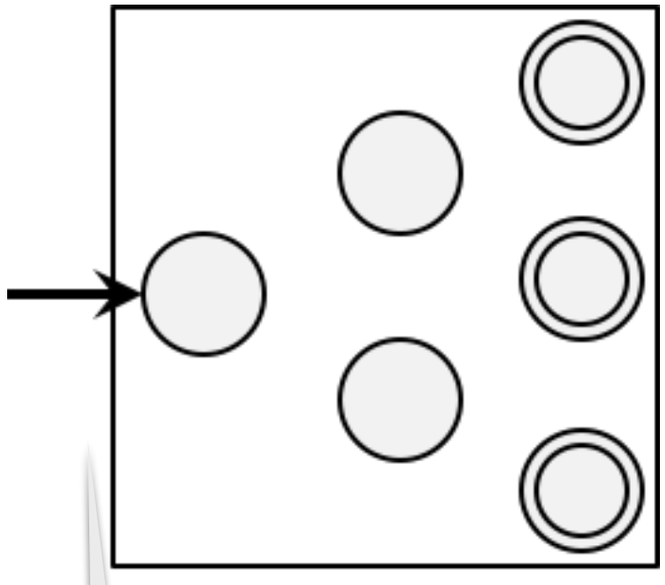
$$A^* = \{w_1 w_2 \cdots w_k \mid k \geq 0 \text{ and each } w_i \in A\}$$

- **Example**. Suppose $L_1 = \{01, 11\}$, what is L^*?

- **Question**. Are regular languages closed under Kleene star?

Kleene Star

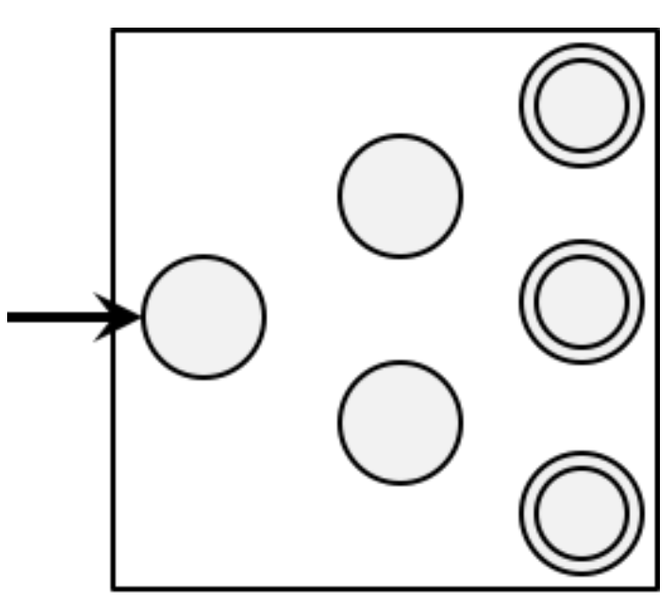
- **Theorem.** The class of regular languages is closed under Kleene star.

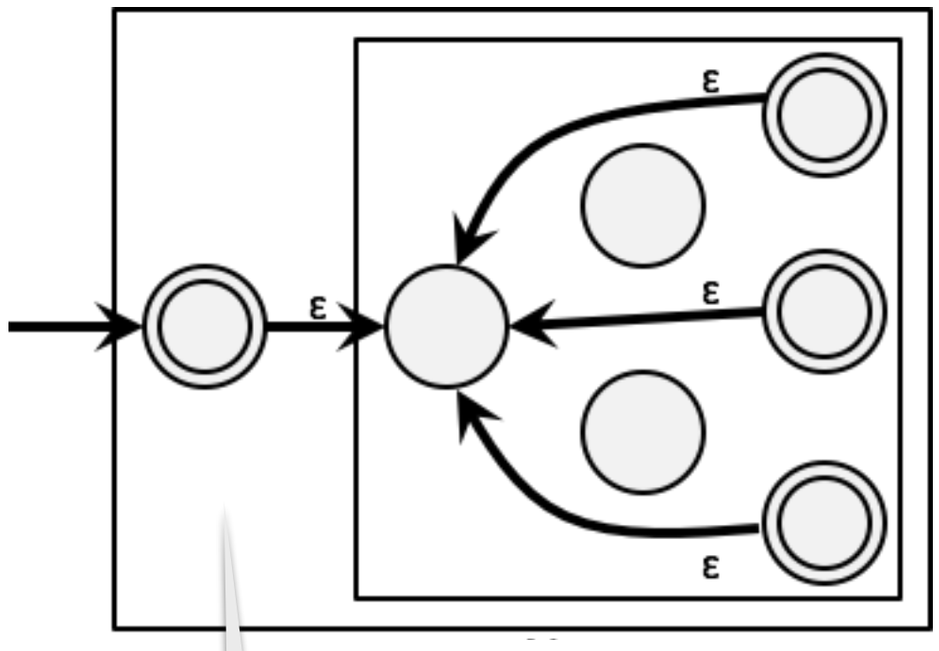


Suppose this is the NFA for L. How to draw an NFA for L^*?

Kleene Star

- **Theorem.** The class of regular languages is closed under Kleene star.





Do we need this new state? Why?

Closed Under Kleene Star

- **Theorem**. The class of languages are closed under Kleene star.
- Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be the NFA for L_1
- Construct NFA $N = (Q, \Sigma, \delta, q_0, F)$ to recognize L_1^*
  - $Q = Q_1 \cup \{q_0\}$ (add a new start state)
  - $F = F_1 \cup \{q_0\}$
  - $$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \end{cases}$$

Closed Under Kleene Star

- **Theorem**. The class of languages are closed under Kleene star.
- Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be the NFA for L_1
- Construct NFA $N = (Q, \Sigma, \delta, q_0, F)$ to recognize L_1^*
  - $Q = Q_1 \cup \{q_0\}$ (add a new start state)
  - $F = F_1 \cup \{q_0\}$
  - $$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \end{cases}$$

Closed Under Kleene Star

- **Theorem**. The class of languages are closed under Kleene star.
- Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be the NFA for L_1
- Construct NFA $N = (Q, \Sigma, \delta, q_0, F)$ to recognize L_1^*
  - $Q = Q_1 \cup \{q_0\}$ (add a new start state)
  - $F = F_1 \cup \{q_0\}$
  - $$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \end{cases}$$

Closed Under Kleene Star

- **Theorem**. The class of languages are closed under Kleene star.
- Proof. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be the NFA for L_1
- Construct NFA $N = (Q, \Sigma, \delta, q_0, F)$ to recognize L_1^*
  - $Q = Q_1 \cup \{q_0\}$ (add a new start state)
  - $F = F_1 \cup \{q_0\}$
  - $$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon. \end{cases}$$

Regular Expressions

Regular Expressions

- A "generative" way to characterize regular languages: **regular expressions**
- Have many applications in programming languages
  - `grep/ awk` in UNIX
- Define regular expressions and give examples
- Show that regular expressions are equivalent to DFA/NFA

Revisit Formal Definition

A regular expression R over the alphabet Σ is defined inductively as follows. R is regular expression if

- **(base cases)**. R is either a for some $a \in \Sigma$, or an empty string ε or an empty set $\emptyset$
- **(recursive cases using regular operators).** R is the union, concatenation or Kleene star of smaller regular expressions that is, $R = R_1 \cup R_2$ or $R = R_1 \circ R_2$ or $R = R_1^*$ where R_1, R_2 are regular expressions

- Let the language of a regular expression $L(R)$ be the set of strings that can be generated by the regular expression
- Examples: 0^*10^*, $(01)^* \cup (10)^*$, $\Sigma\Sigma$, etc

Working with Regular Expressions

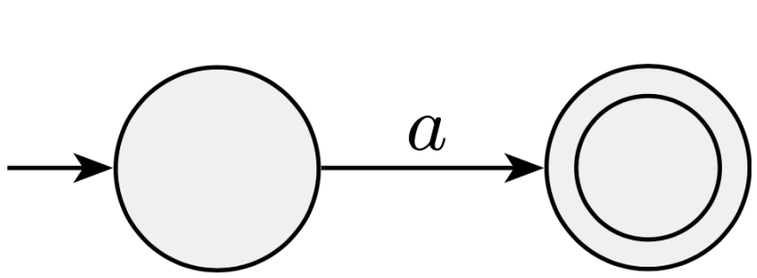
- If R is a regular expression then:
  - $R \cup \emptyset = ?$
  - $R \circ \varepsilon = ?$
- But the following may not hold:
  - $R \cup \varepsilon = ? R$
  - What is $R \circ \emptyset$?

Working with Regular Expressions

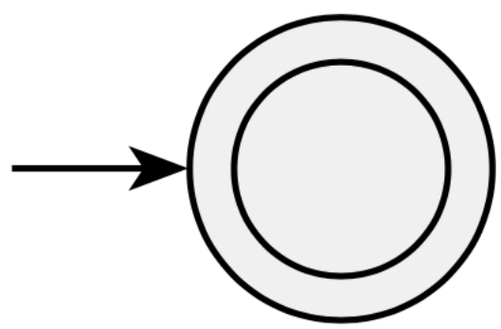
- If R is a regular expression then:
  - $R \cup \emptyset = R$
  - $R \circ \varepsilon = R$
- But the following may not hold:
  - $R \cup \varepsilon$ is not necessarily the same as R
  - $R \circ \emptyset = \emptyset$

Equivalence with Finite Automata

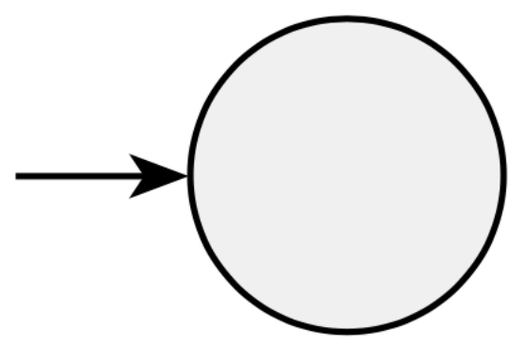
- **Lemma (1.55 in Sipser)**. If a language is described by a regular expression, then is regular
- **Proof.** Let R is the regular expression, sufficient to create an NFA that recognizes $L(R)$
    - (base cases). It is easy to create an NFA for each of the base cases for R



$R = a$



$R = \varepsilon$



$R = \emptyset$

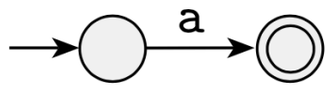
Equivalence with Finite Automata

- **Lemma (1.55 in Sipser)**. If a language is described by a regular expression, then is regular
- **Proof.** Let R is the regular expression, sufficient to create an NFA that recognizes $L(R)$
    - (recursive cases). Suppose by induction we have an NFA for any regular expression smaller than R, we can create an NFA for R using the union/concatenation/Kleene star of these NFA

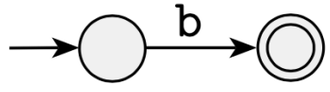
Regular Expression to NFA: Example 1.56

- Convert regular expression (ab ∪ a*) to an NFA

a



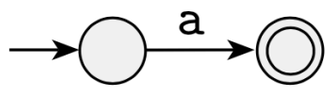
b



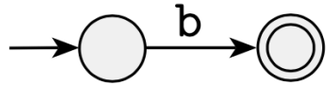
Regular Expression to NFA: Example 1.56

- Convert regular expression (ab ∪ a*) to an NFA

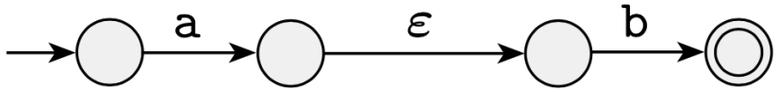
a



b



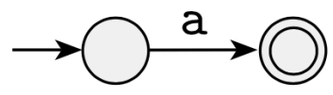
ab



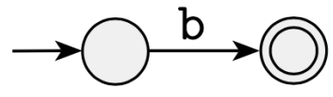
Regular Expression to NFA: Example 1.56

- Convert regular expression (ab ∪ a*) to an NFA

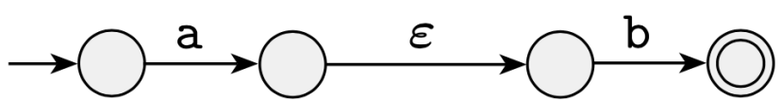
a



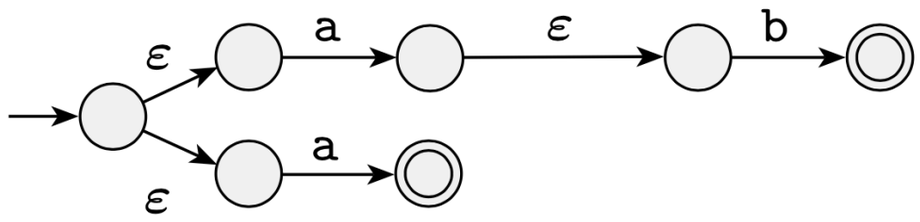
b



ab



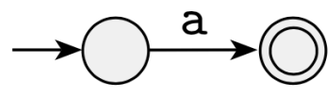
ab ∪ a



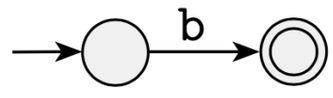
Regular Expression to NFA: Example 1.56

- Convert regular expression (ab ∪ a*) to an NFA

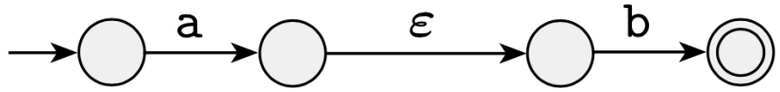
a



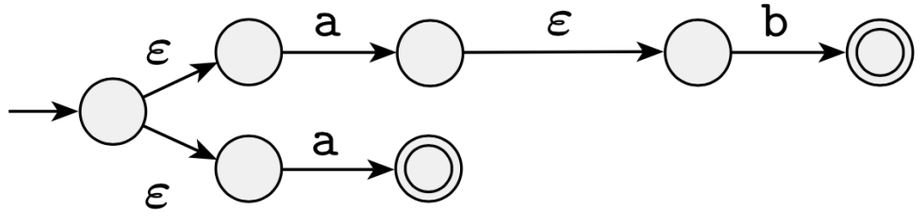
b



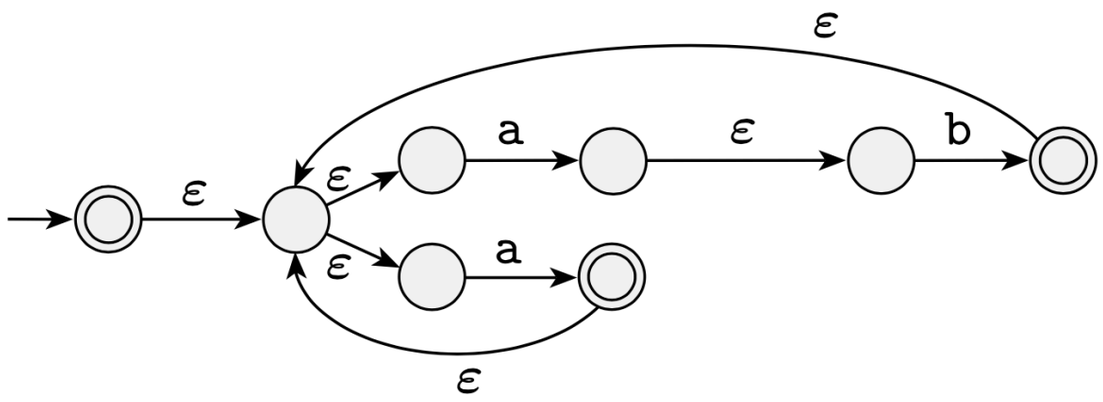
ab



ab ∪ a



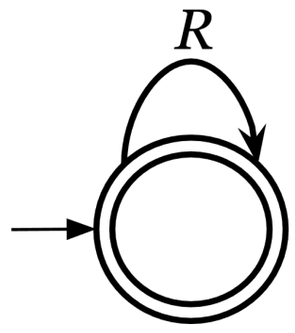
(ab ∪ a)*



NFAs to Regular Expression

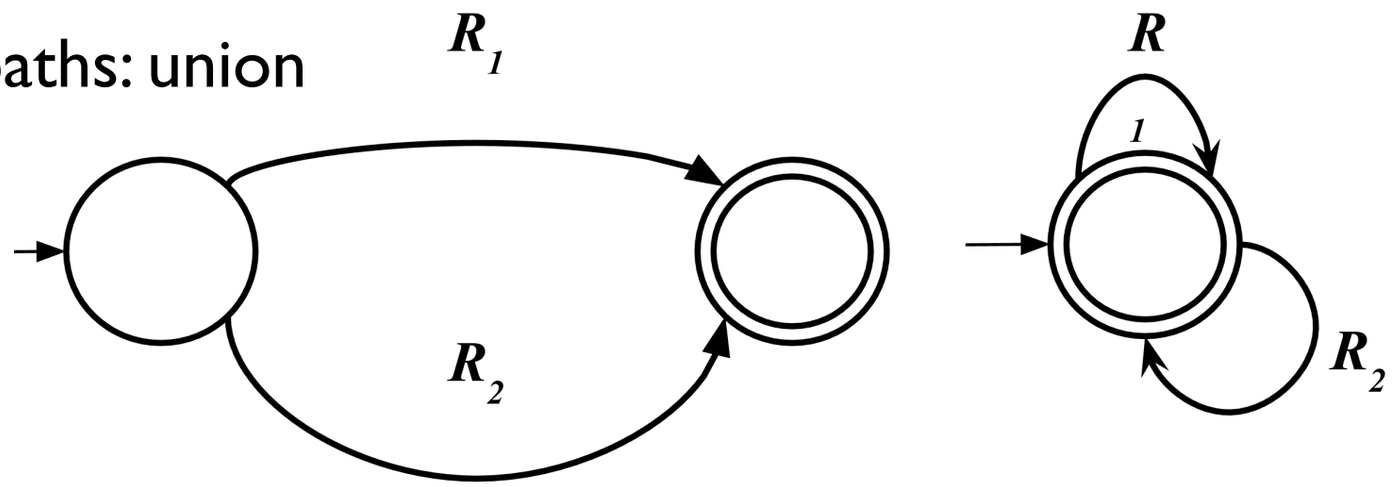
- Every NFA can be converted to an equivalent regular expression

Self loops: Kleene star

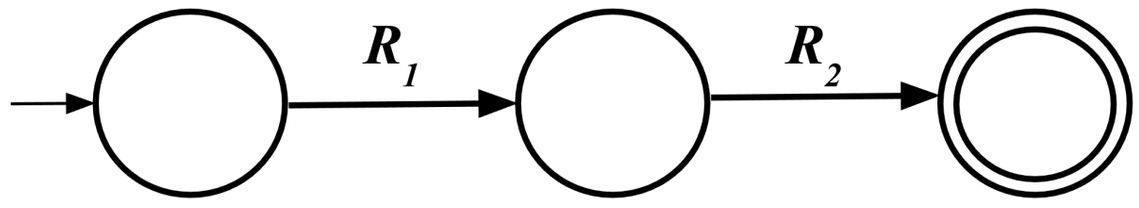


R^*

Alternate paths: union



$R_1 \cup R_2$



$R_1 \circ R_2$

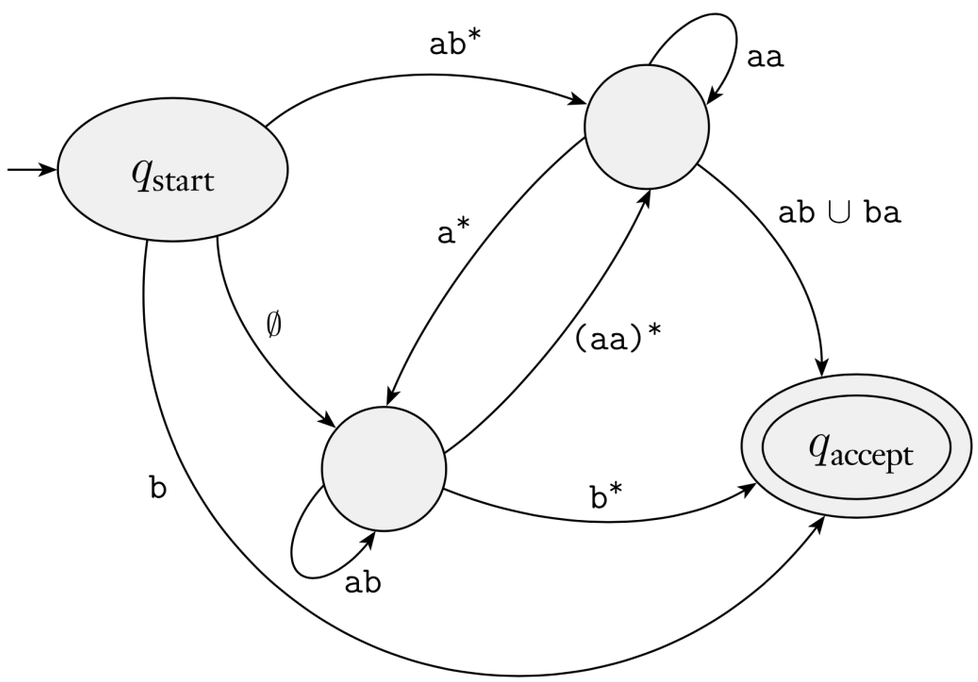
Adjacent paths: concatenation

Converting a DFA to Regular Expression

- **Lemma (1.60 in Sipser)**. If a language is regular (recognized by a DFA), then it can be described by some regular expression.
- **Proof outline.**
  - Convert the DFA into a GNFA (generalized NFA) with $k \geq 2$ states (NFA where transitions occur on regular expressions)
  - Eliminate states of the GNFA one by one until two states left
  - Output the regular expression from the start to accept state

Generalized NFA

- A GNFA is a generalized NFA with the following conditions:
  - Transitions are on regular expressions (not just symbols or ε)
  - Start state has an arrow to every other state and not arrows coming in from any state
  - Only one accept state that has arrows coming in from every other state and no arrows leaving it
  - Every other state has arrows to every other state including itself

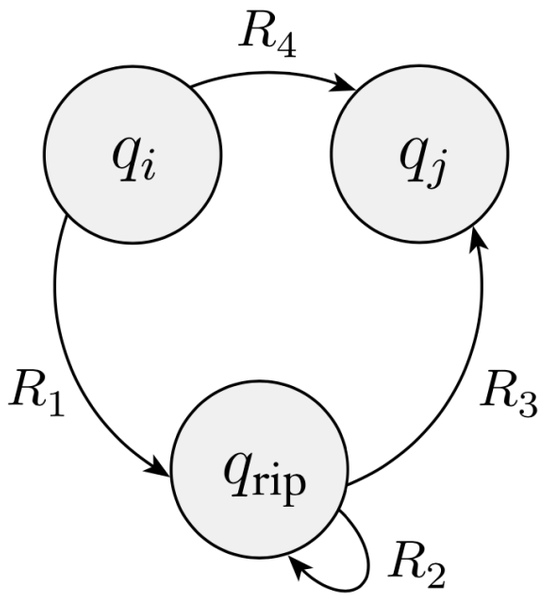


DFA $\Longrightarrow$ GNFA $\Longrightarrow$ RegularExp

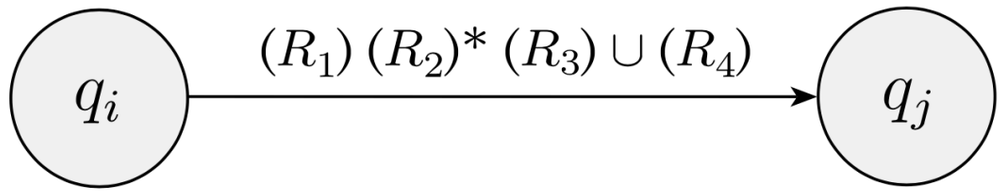
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, we can convert it to a regular expression as follows
- GNFA G: add a new start state q_s and accept state q_f to M
    - If there is an arrow missing from q_s to a state $q \in Q$, add an arrow labelled with $\emptyset$
        - Add ε arrows from F to q_f
- For any pair of states (p, q) that are neither start or accept states of M, add additional $\emptyset$ arrows to create a valid GNFA
- Now perform the state-elimination algorithm described next to convert G with k states to G with $k - 1$ states

State Elimination Algorithm

- Consider a GNFA with $k > 2$ states and let q_{rip} be a state that is neither the start or accept state
- Reduce: create a GNFA with $k - 1$ states by removing q_{rip}; replace all paths that go through it with an equivalent regular expression



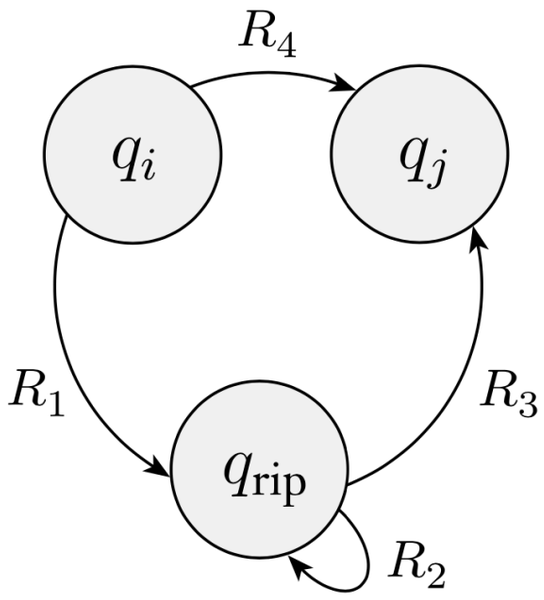
before



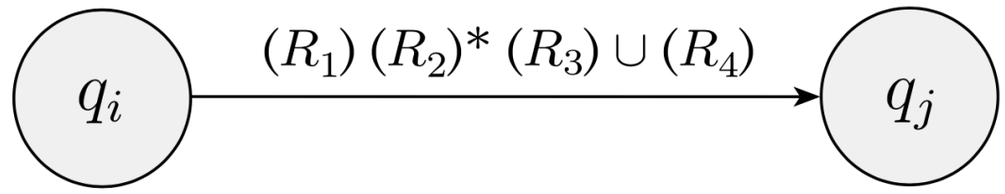
after

Final Regular Expression

- Perform the state elimination algorithm until there are $k = 2$ states (start and accept) states left
- Output the regular expression on the only remaining transition
- **Correctness:** by induction on the number of states of GNFA

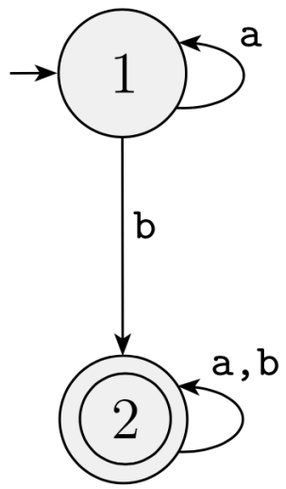


before



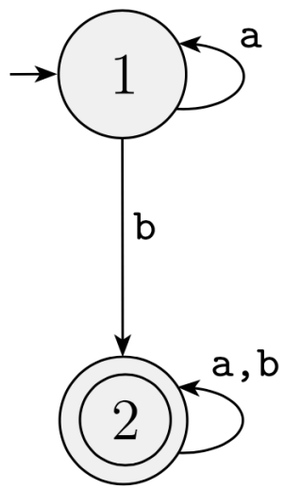
after

DFA to Regular Expression Example

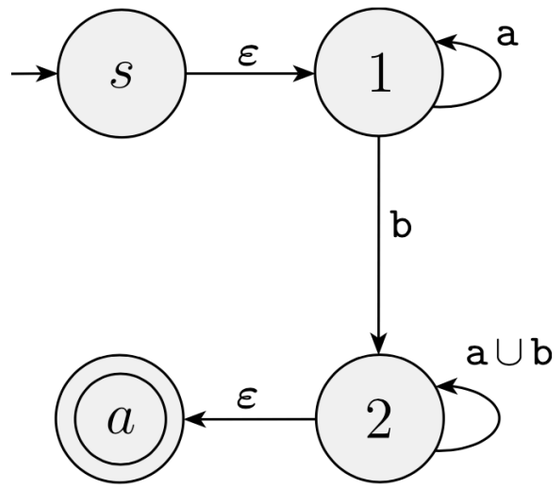


(a)

DFA to Regular Expression Example

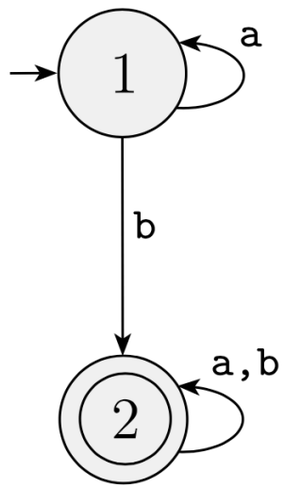


(a)

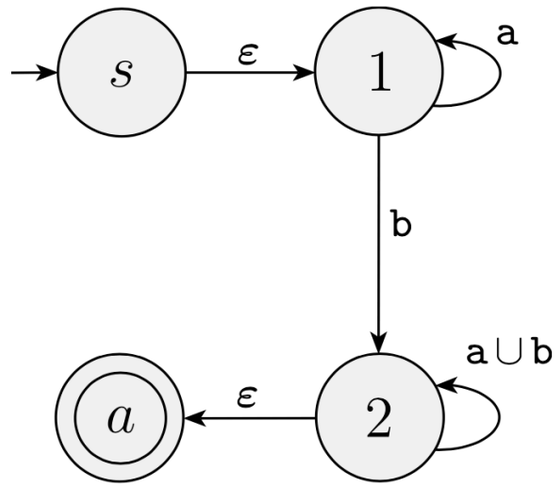


(b)

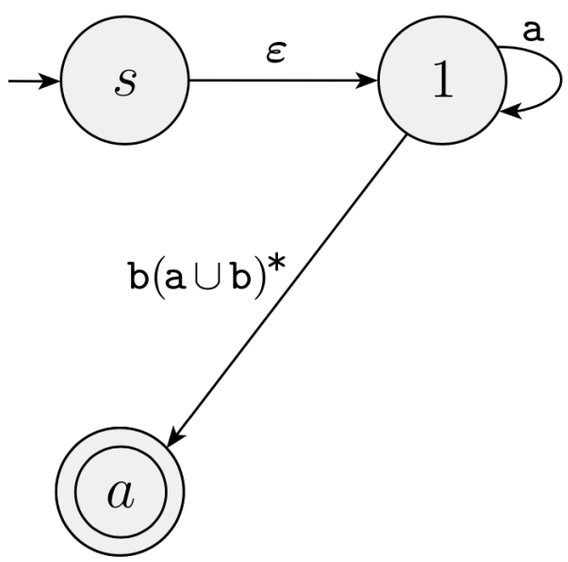
DFA to Regular Expression Example



(a)

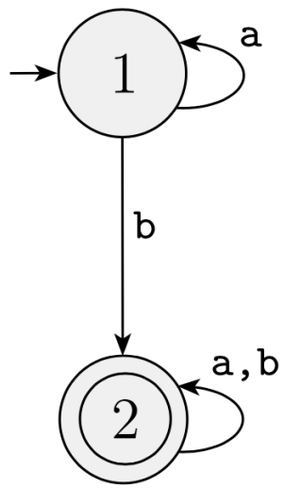


(b)

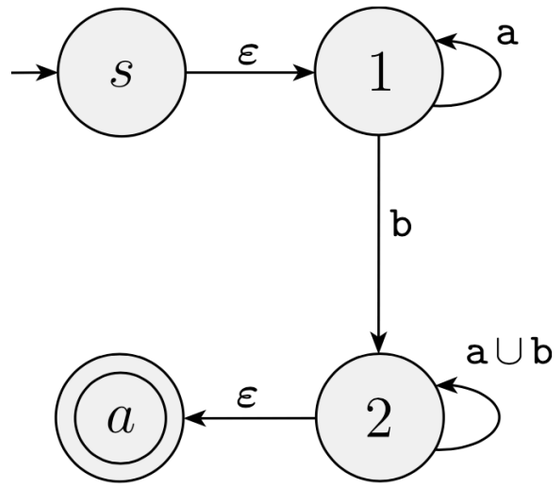


(c)

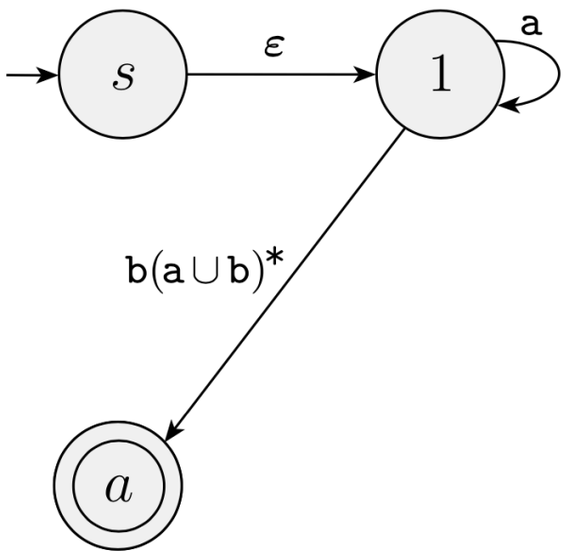
DFA to Regular Expression Example



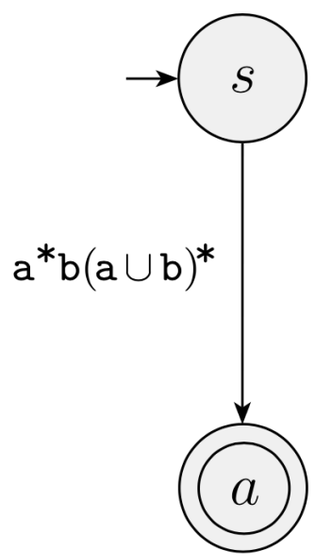
(a)



(b)



(c)



(d)

DFA $\Longrightarrow$ GNFA $\Longrightarrow$ RegularExp

- Why does this reduction work?
- **Claim.** Consider a GNFA G with k states, let G' be GNFA after the state-elimination algorithm is performed once, then both G and G' accept the same language.
    - By construction

- **Corollary.** The language of the regular expression constructed from this algorithm is the same as the starting DFA

Takeaways

- Regular expressions provide an alternate "generative" way to describe regular languages
- Three ways to characterize regular languages:
    - DFAs
    - NFAs
    - Regular languages

Not All Languages Are Regular

- Any language does not a DFA that recognizes it is not-regular
- How we do ***prove no such DFA*** exists?
    - First example of an impossibility results in this class
    - Many more to come

- Intuitively, any decision problem that requires ***finite*** memory "to solve" is regular
- **Question.** Are finite languages regular?
    - $L \subseteq \Sigma^*$ and $|L|$ is finite
    - All finite languages are regular

All Finite Languages are Regular

- **Theorem.** All finite languages are regular.
- $L = \{w_1, \ldots, w_n\}$ for some $n \in \mathbb{N}$
- Let $L_i = \{w_i\}$ for each $i \in \{1,\ldots,n\}$
- $L = \cup_{i=1}^{n} L_i$

- **Claim 1.** Each L_i is regular.

- **Claim 2.** A finite union of regular languages is regular.

- Using Claim 1 and 2, L is regular

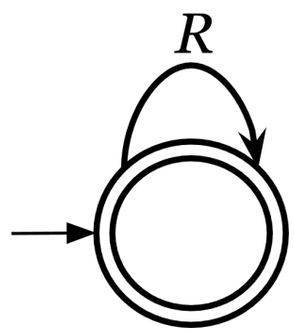
Infinite Regular Languages

- Have seen many infinite regular languages
- What do they have in common?

Structure of Infinite Regular Languages

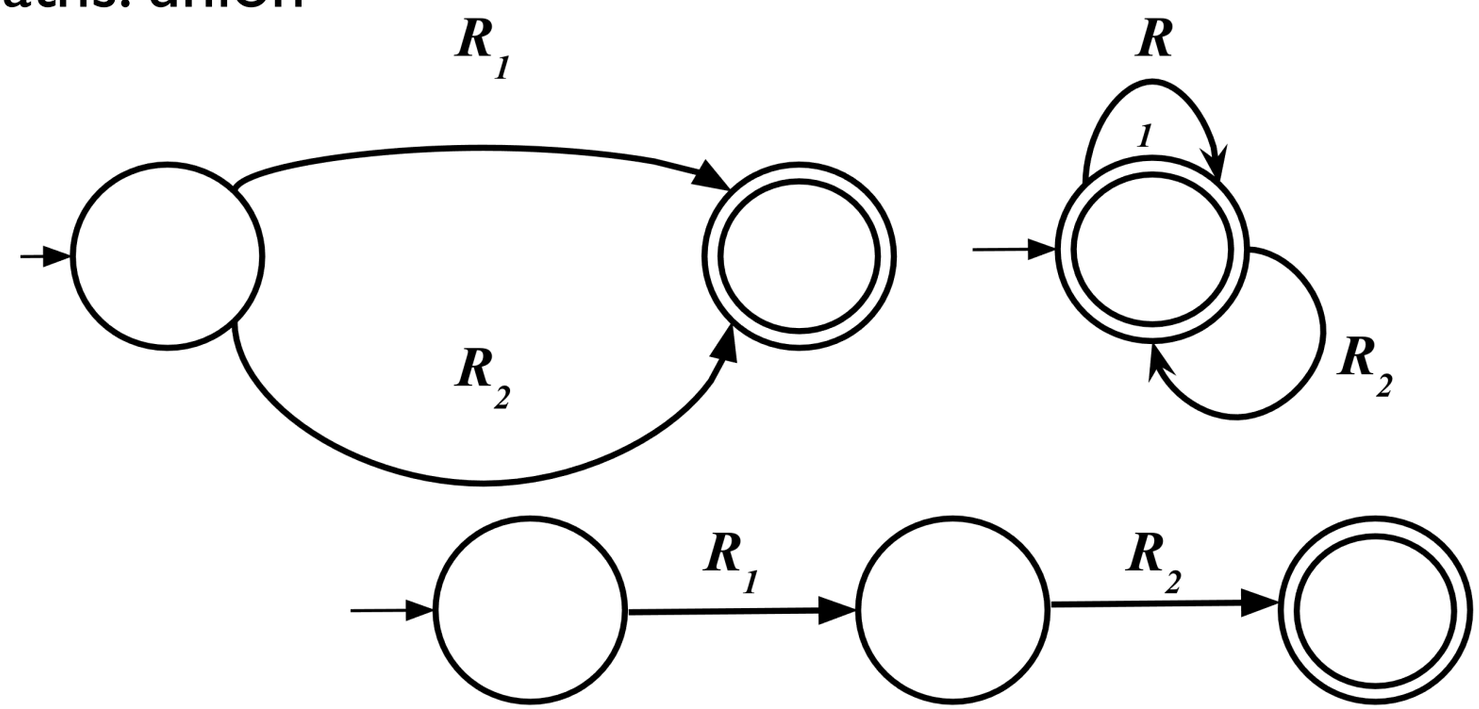
- Which of these are responsible for going from finite to infinite?

Self loops: Kleene star



R^*

Alternate paths: union



$R_1 \cup R_2$

$R_1 \circ R_2$

Adjacent paths: concatenation

Loops in DFA: Intuition

- Consider the DFA $M's$ transitions on an input string w
- It enters some states $q_0, \ldots, q_1, q_2, \ldots, q_n$
- **Question.** If there is a "loop" what does that mean about the states visited?

- Now suppose two different strings x, y bring M to the same state q
- Consider attaching the same suffix z to both
- **Question.** What can we say about the state M is in after reading input string xz versus after reading input string yz?

Indistinguishability (DFA)

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Let x, y be any string over Σ.

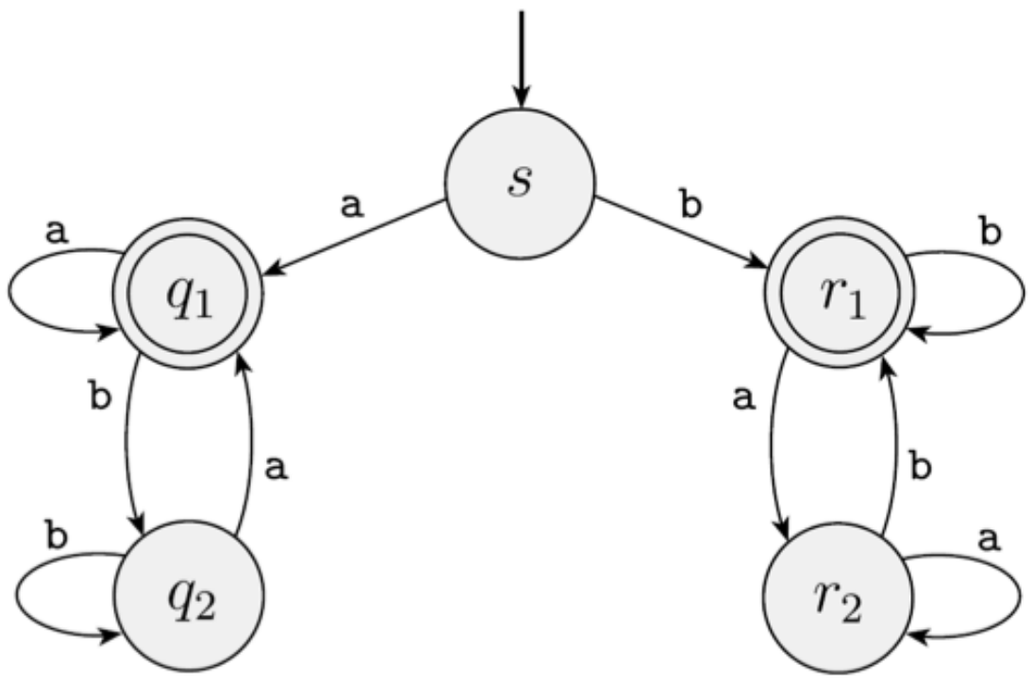
Definition. x **indistinguishable to** y with respect to a DFA M, denoted $x \sim_M y$ if and only if $\delta^*(q_0, x) = \delta^*(q_0, y)$ (i.e., the state reached by M on x is the same as the state reached by M on y)

Corollary. If $x \sim_M y$ then for all $z \in \Sigma^*$, then

$xz \in L(M) \iff yz \in L(M)$

Class Exercise

- **Example.** $L = \{w \in \{a, b\}^* \mid w$ starts and ends with the same symbol$\}$
- **Definition.** x **indistinguishable to** y with respect to a DFA M, denoted $x \sim_M y$ if and only if $\delta^*(q_0, x) = \delta^*(q_0, y)$ (i.e., the state reached by M on x is the same as the state reached by M on y)
- **Question:** for each state in the DFA for L, write a regular expression characterizing all strings that bring the DFA to that state.



Indistinguishability (Languages)

Let L be any language over an alphabet Σ.

Definition. x **indistinguishable to** y with respect to L, denoted $x \equiv_L y$ if and only if for all $z \in \Sigma^*$, we have that $xz \in L \iff yz \in L$

Observation: $\equiv_L$ is an equivalence relation over Σ^*

Thus, $\equiv_L$ ***partitions*** Σ^* into equivalence classes.

Distinguishing Suffixes

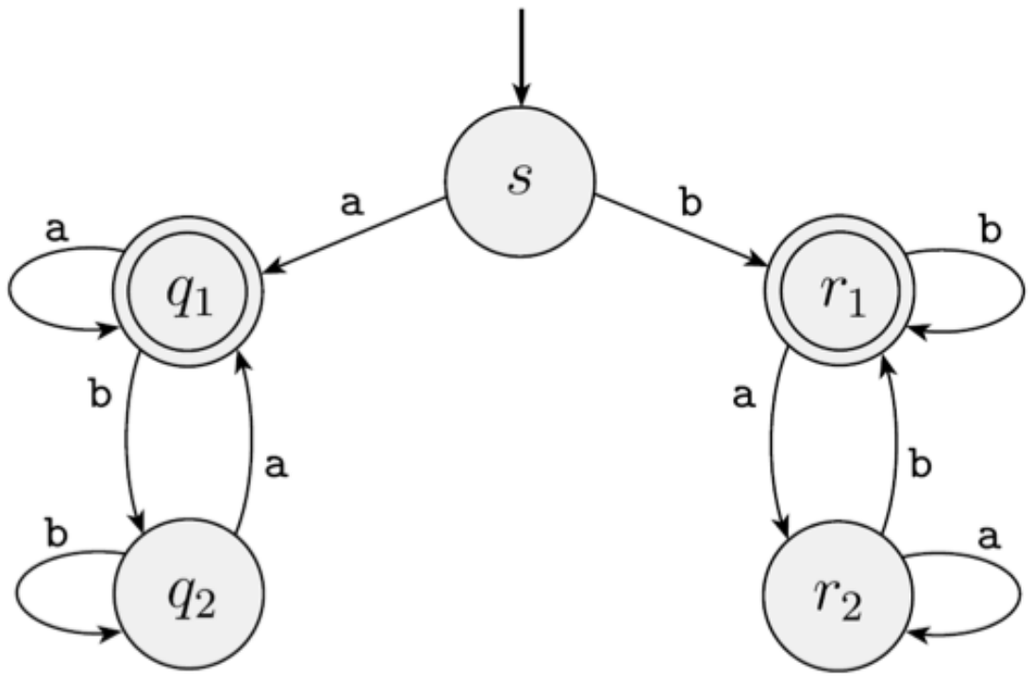
- Every string in the same equivalence class $[x]$ of $\equiv_L$ are indistinguishable with each other
- Two strings $x, y \in \Sigma^*$ are in different equivalence iff they are ***distinguishable***
  - Can find a suffix $z \in \Sigma^*$ that distinguishes them, that is, $xz \in L$ and $yz \notin L$ or $xz \notin L$ and $yz \in L$
- **Question.** Suppose $x \in L$ and $y \notin L$, are they distinguishable?

Indistinguishability (Languages)

- **Example.**

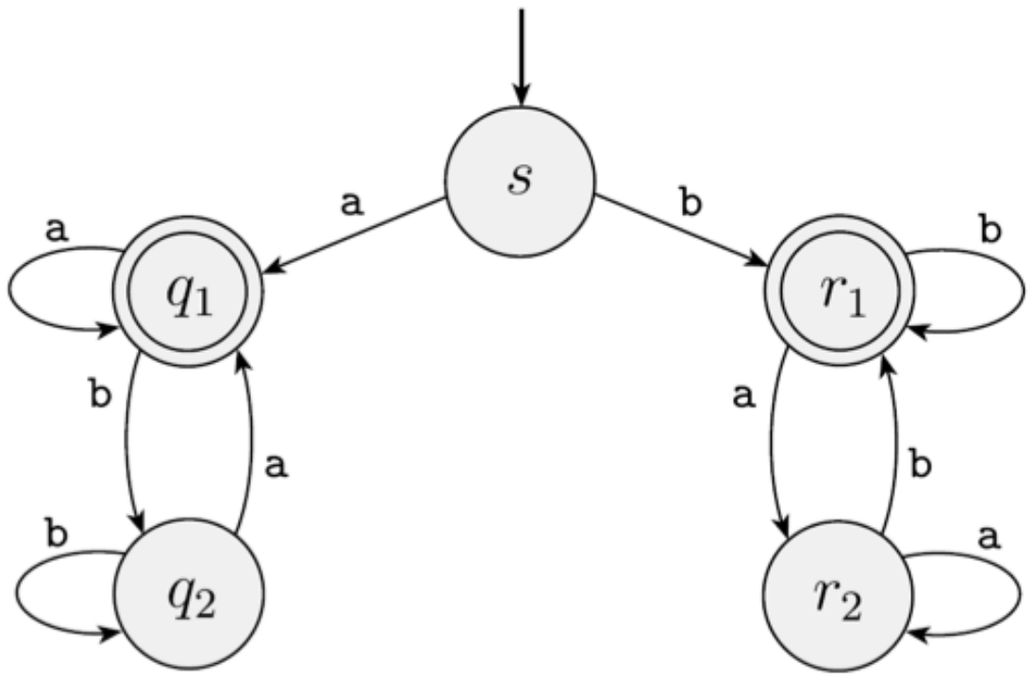
  $L = \{w \in \{a, b\}^* \mid w \text{ starts and ends with the same symbol}\}$

- **Problem.** Find the equivalence classes of the relation $\equiv_L$.



Indistinguishability DFA vs Languages

- **Claim.** If $x \sim_M y$, then $x \equiv_{L(M)} y$.



Minimal DFA

- **Claim.** If a language L over Σ has k equivalence classes defined by $\equiv_L$, then any DFA for L must have at least k states.
- **Corollary.** If a DFA M for L has number of states equal to the number of equivalence classes of $\equiv_L$ then such a DFA is minimal.

Myhill-Nerode Theorem

Let L be a language over Σ^*, then L is regular **if and only if** the relation $\equiv_L$ over Σ^* has a finite number of equivalence classes.

Myhill-Nerode Theorem

Let L be a language over Σ^*, then L is regular **if and only if** the relation $\equiv_L$ over Σ^* has a finite number of equivalence classes.

Necessary condition. For L to be regular, it must have finitely many equivalence classes. Equivalently, if $\equiv_L$ over Σ^* has an infinite number of equivalence classes, then L cannot be regular.

Sufficient condition. If $\equiv_L$ has finitely many equivalence classes, then L must be regular.

Proving Non Regularity

- Myhill-Nerode theorem says that any language that has infinitely many equivalence classes with respect to $\equiv_L$ is not regular
- Typically, we don't need to find all of equivalence classes
- Sufficient to find an infinite subset of strings that are mutually distinguishable

Fooling Sets

Definition. A set of strings $S \subseteq \Sigma^*$ is a **fooling set** with respect to a language $L \subseteq \Sigma^*$ if every pair of strings in S is distinguishable with respect to each other.

Example. $L = \{w \in \{a, b\}^* \mid w$ starts and ends with the same symbol$\}$

An example fooling set for L?

Question. Can the size of a fooling set be bigger than the number of equivalence classes?

- Max size of a fooling set for L = # of equivalence class of $\equiv_L$
- Size of any fooling set for $L \leq$ # of equivalence class of $\equiv_L$

Myhill-Nerode Theorem

Maximum fooling set size of L

$=$ # equivalence classes of $\equiv_L$

$=$ minimum states of DFA for L

Takeaway. If we could prove that there exists an infinite number of distinguishable sets for a language, it would mean that even the smallest DFA for the language would require an infinite number of states. Therefore, no such DFA exists and the language cannot be regular.

Proving Non-Regularity

Problem. Prove that the language $L = \{a^i b^i \mid i \in \mathbb{N}\}$ is not regular.

Hint. Identify and prove that L has an infinite fooling set.

Proving Non-Regularity

Problem. Prove that the langueg

$L = \{a^n \mid n \in \mathbb{N} \text{ and } n \text{ is a power of 2}\}$ is not regular.

Hint. Identify and prove that L has an infinite fooling set.