CSCI 361 Lecture 5:

Regular Expressions

Shikha Singh

Announcements & Logistics

- **No class today (Sept 19)!**
  - I am at Tapia Celebration of Diversity in Computing (San Diego)
- In lieu of a lecture, please refer to the textbook (Chapter 1.3)
- These slides break these readings down and state main takeaways
- Questions in HW 2 is based on these concepts
- We will move on to "Non regular" languages in Lecture 6 Sept 24

Overview So Far

- Showed the equivalence of DFAs and NFAs
- Regular languages are those that are recognized by a DFA/ NFA
- Regular languages are closed under:
    - Complement
    - Union
    - Intersection
    - Concatenation
    - Kleene star

Today

- 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

Formal Definition: Regular Expression

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

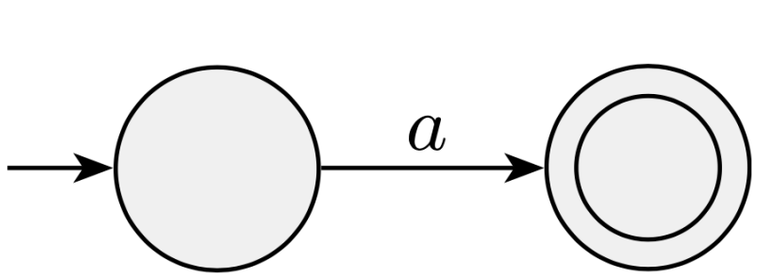
- Let the language of a regular expression $L(R)$ be the set of strings that can be generated by the regular expression
- Examples: $0*10*$, $(01)* \cup (10)*$, $\Sigma\Sigma$, etc

Identities of Regular Expressions

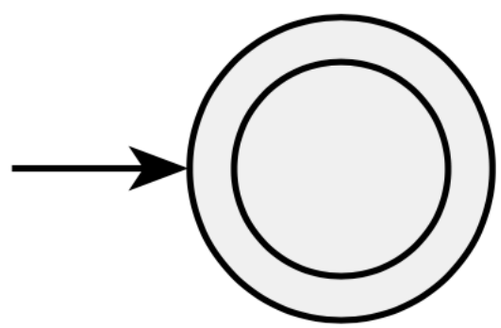
- If R is a regular expression then:
  - $R \cup \emptyset = R$
  - $R \circ \varepsilon = R$
- But the following may not hold:
  - $R \cup \varepsilon$ not may be same as R
  - $R \circ \emptyset$ is $\emptyset$ (and not same as R)

Equivalence with Finite Automata

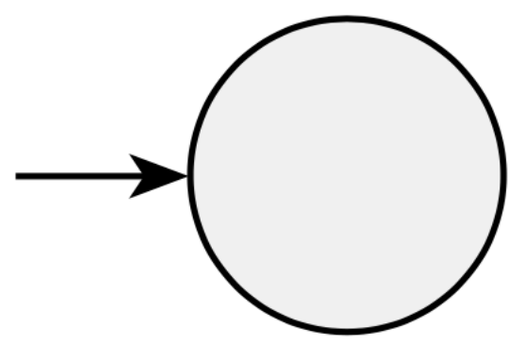
- **Lemma (1.55 in Sipser)**. If a language is described by a regular expression, then is regular
- **Proof.** Let R is the regular expression, sufficient to create an NFA that recognizes $L(R)$
    - (base cases). It is easy to create an NFA for each of the base cases for R, see below



$R = a$



$R = \varepsilon$



$R = \emptyset$

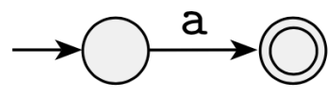
Equivalence with Finite Automata

- **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

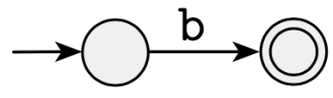
Regular Expression to NFA: Example 1.56

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

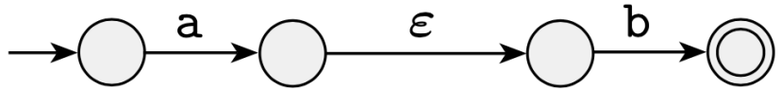
a



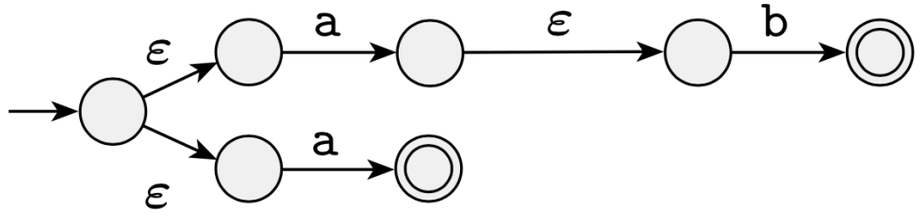
b



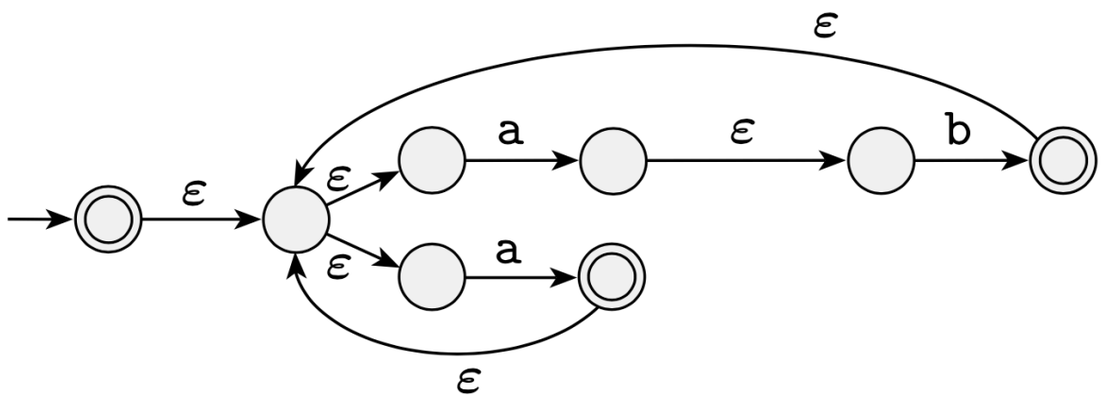
ab



ab ∪ a



(ab ∪ a)*

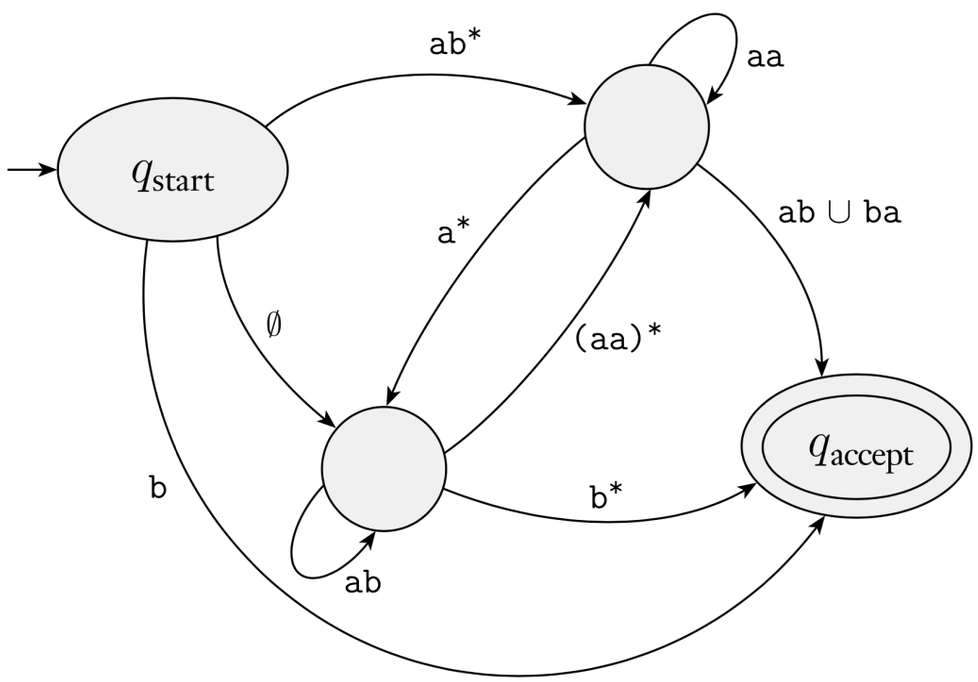


Converting a DFA to Regular Expression

- **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 (defined on next slide)
  - 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

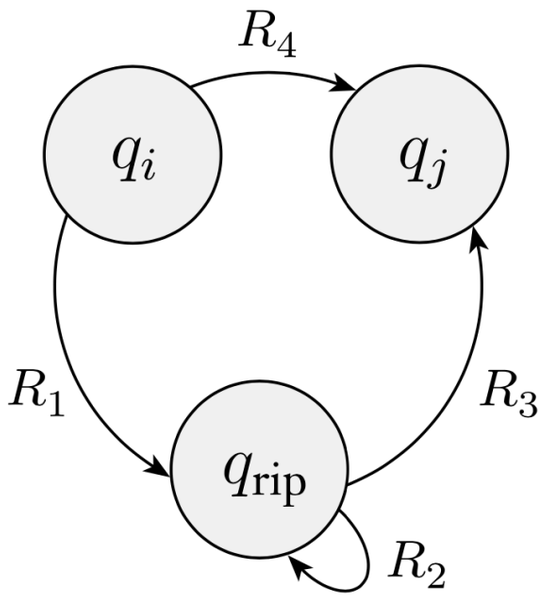


DFA $\Longrightarrow$ GNFA $\Longrightarrow$ RegularExp

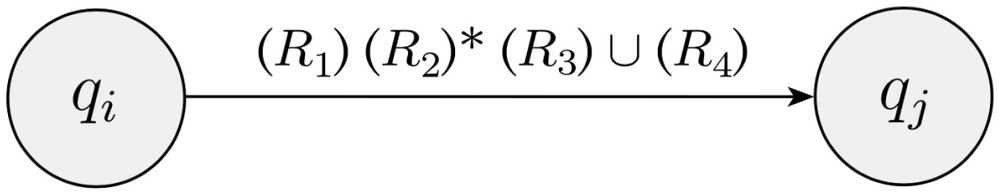
- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA, we can convert it to a regular expression as follows
- GNFA G: add a new start state q_s and accept state q_f to M
    - If there is an arrow missing from q_s to a state $q \in Q$, add an arrow labelled with $\emptyset$
        - Add ε arrows from F to 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
- Create a GNFA with $k - 1$ states by removing q_{rip} and replacing all transition paths that went through it with an equivalent regular expression



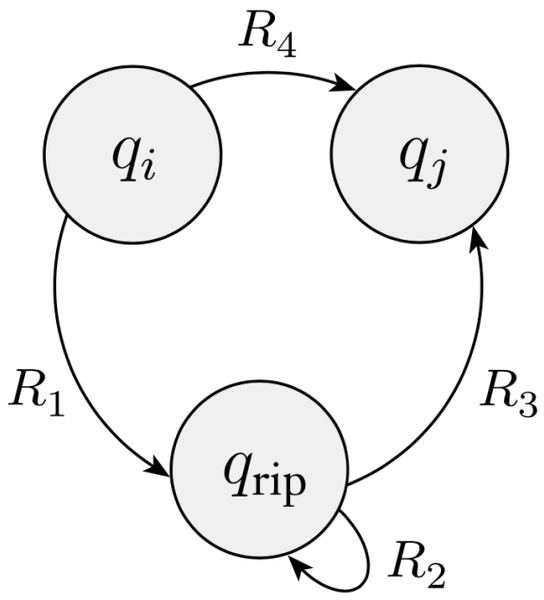
before



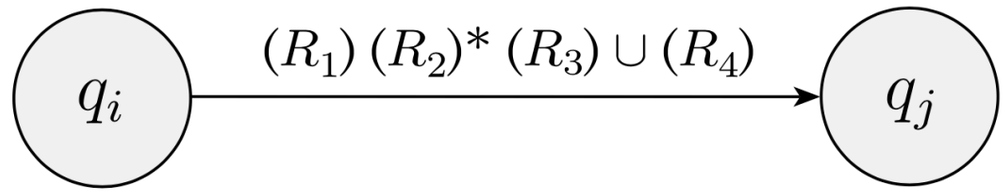
after

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

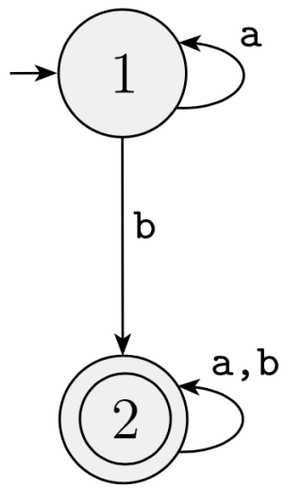


before

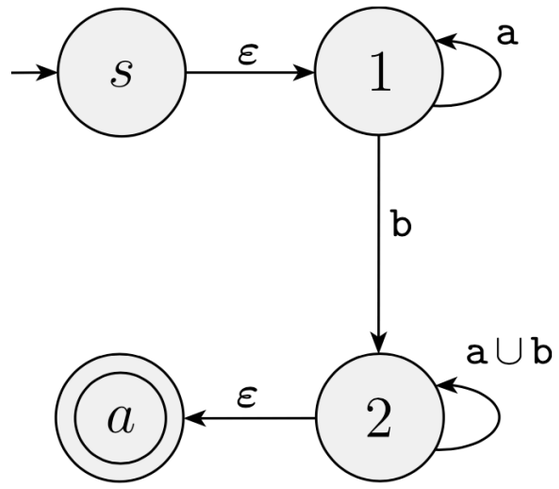


after

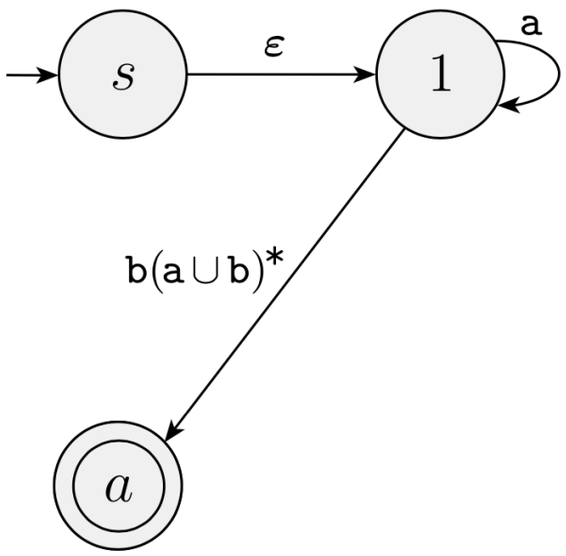
DFA to Regular Expression Example



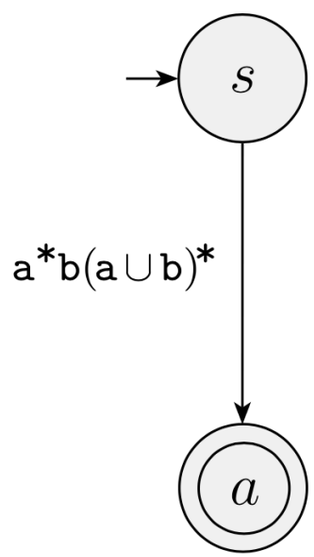
(a)



(b)



(c)



(d)

Takeaways

- Regular expressions provide an alternate "generative" way to describe regular languages
- Three ways to characterize regular languages:
    - DFAs
    - NFAs
    - Regular languages
- Next time: non-regular languages
    - How do we identify and prove that languages are not regular?
    - Identify the minimum # of states required by a DFA