CSC 320
Foundations of Computer Science

Lecture 5

Instructor: Dr. Ulrike Stege

Territory Acknowledgement

We acknowledge and respect the lək̓ʷəŋən peoples on whose traditional territory the university stands and the Songhees, Esquimalt and W̱SÁNEĆ peoples whose historical relationships with the land continue to this day.

This meeting will be recorded

"Please be aware our sessions are being screen-recorded to allow students who are not able to attend to watch later and will be posted in Brightspace."

Deadlines; Assessment

10%

Quizzes

Quiz 1-8: 1% each
Quiz 9: 2%

25%

Assignments

Assignment 1-5: 5% each

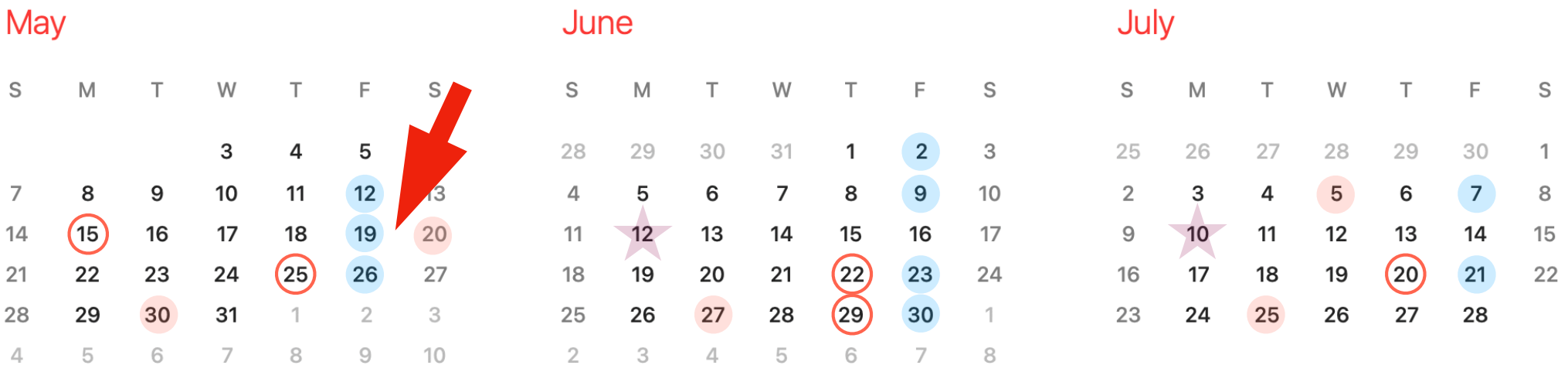
25%

Midterms

Midterm 1: 10%
Midterm 2: 15%

40%

Final Exam



Timed quizzes (~30 min)
Review before starting quiz

Last time ….

- NFAs
- Equivalence of NFAs and DFAS
- Regular languages are closed under concatenation
- Regular expressions
- Languages of regular expressions

Regular Languages: Equivalences

- The set of regular languages is
  - the set of all languages recognized by deterministic finite automata, and also it is the same as
  - the set of all languages recognized by nondeterministic finite automata, as well as it is the same as

Today …

- The languages of regular expressions are exactly the regular languages
- DFA state minimization

Before we start:

- Given a finite automaton (DFA or NFA) with more than one accept state, how can we build a finite automaton that recognizes the same language but has exactly one accept state?

- What type of finite automaton (DFA or NFA) is this?

Summary: Regular Expressions

- Defined inductively. Regular expressions are:
  - a ($a \in \Sigma$), ϵ, $\varnothing$
  - $(R_1 \cup R_2)$, $(R_1 R_2)$ and $(R_1{}^*)$, where R_1 and R_2 are regular expressions
- Parenthesis can be omitted
  - If no parentheses evaluate in order: star, concatenation, union
- Note:
  - $R^+ := RR^*$
  - $R \cup \varnothing = R$
  - $R\epsilon = R$
  - $\epsilon R = R$
  - $R\varnothing = \varnothing$
  - $\varnothing R = \varnothing$

Definition: *Language* recognized by a regular expression

Assume R_1 and R_2 are regular expressions. The **language** $L(R)$ **for regular expression** R is defined as:

- If $R = a$, for some $a \in \Sigma$, then $L(R) = \{a\}$
- If $R = \epsilon$ then $L(R) = \{\epsilon\}$
- If $R = \emptyset$ then $L(R) = \emptyset$
- If $R = (R_1 \cup R_2)$ then $L(R) = L(R_1) \cup L(R_2)$
- If $R = (R_1 R_2)$ then $L(R) = L(R_1)L(R_2)$
- If $R = (R_1^*)$ then $L(R) = L(R_1)^*$

Equivalence of regular languages and the set of languages of regular expressions

Theorem: A language is regular if and only if there exists some regular expression that describes it

Proof consists of two parts

1. If a language is described by a regular expression, then it is regular

2. If a language is regular, then it can be described by a regular expression

Claim 1. If a language is described by a regular expression, then it is regular

- We show: given a regular expression R, there exists a finite automaton M with $L(M) = L(R)$

- Proof by induction; we distinguish the following cases:

1. $R = a$, $a \in \Sigma$

2. $R = \epsilon$

3. $R = \emptyset$

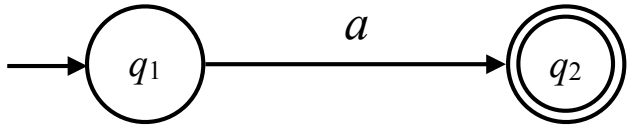
4. $R = (R_1 \cup R_2)$, where R_1, R_2 are regular expressions

5. $R = (R_1 R_2)$, where R_1, R_2 are regular expressions

6. $R = (R_1^*)$ where R_1 is a regular expression

Base Case 1: $R = a$, $a \in \Sigma$
Show: $L(a)$ is regular

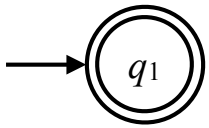
- $L(a) = \{a\}$
- NFA $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$ with



  - $\delta(q_1, a) = \{q_2\}$
  - $\delta(r, b) = \emptyset$ for $(r, b) \neq (q_1, a)$, $b \in \Sigma \cup \{\epsilon\}$

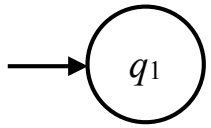
Base Case 2: $R = \epsilon$
Show: $L(R)$ is regular

- $L(\epsilon) = \{\epsilon\}$
- NFA $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$ with
  - $\delta(q_1, b) = \emptyset$ for any $b \in \Sigma \cup \{\epsilon\}$



Base Case 3: $R = \emptyset$
Show: $L(R)$ is regular

- $L(\emptyset) = \emptyset$
- NFA $N = (\{q_1\}, \Sigma, \delta, q_1, \emptyset)$ with



  - $\delta(q_1, b) = \emptyset$ for any $b \in \Sigma \cup \{\epsilon\}$

Induction Step

- Proof by induction; we distinguish the following cases:

1. $R = a,\ a \in \Sigma$ ✓
2. $R = \epsilon$ ✓
3. $R = \emptyset$ ✓

Base cases: $L(R)$ is regular

4. $R = (R_1 \cup R_2)$, where R_1, R_2 are regular expressions
5. $R = (R_1 R_2)$, where R_1, R_2 are regular expressions
6. $R = (R_1^*)$ where R_1 is a regular expression

Show: $L(R)$ is regular

Case 4: $L(R_1 \cup R_2)$ is regular

- Given regular expressions R_1 and R_2 where we assume that $L(R_1)$ and $L(R_2)$ are regular languages
- From definition of languages recognized by regular expressions (Case 4): $L(R_1 \cup R_2) = L(R_1) \cup L\ (R_2)$
- We know: regular languages closed under union
  - Therefore: $L(R_1) \cup L(R_2)$ is a regular language
- Therefore: $L(R_1 \cup R_2)$ is regular

Case 5: $L(R_1 R_2)$ is regular

- Given regular expressions R_1 and R_2 where we assume that $L(R_1)$ and $L(R_2)$ are regular languages
- From definition of languages recognized by regular expressions (Case 5): $L(R_1 R_2) = L(R_1)\ L(R_2)$
- We know: regular languages closed under concatenation
  - Therefore $L(R_1)\ L(R_2)$ is regular
- Therefore, $L(R_1 R_2)$ is regular

Case 6: $L(R_1{}^*)$ is regular

- Given regular expression R_1 where we assume that $L(R_1)$ is a regular language
- From definition of languages recognized by regular expressions: $L(R_1)^* = L(R_1{}^*)$
- We know: the set of regular languages closed under star
  - Therefore: $L(R_1)^*$ is regular
- Thus: language $L(R_1)^*$ described by regular expression $R_1{}^*$ is regular

Equivalence of regular languages and the set of languages of regular expressions

Theorem: A language is regular if and only if there exists some regular expression that describes it

Proof consists of two parts

1. If a language is described by a regular expression, then it is regular

2. If a language is regular, then it can be described by a regular expression

Induction Step

- Proof by induction; we distinguish the following cases:

1. $R = a,\ a \in \Sigma$ ✓
2. $R = \epsilon$ ✓
3. $R = \emptyset$ ✓

Base cases: $L(R)$ is regular

4. $R = (R_1 \cup R_2)$, where R_1, R_2 are regular expressions ✓
5. $R = (R_1 R_2)$, where R_1, R_2 are regular expressions ✓
6. $R = (R_1^*)$ where R_1 is a regular expression ✓

Inductive cases: $L(R)$ is regular

Equivalence of regular languages and the set of languages of regular expressions

Theorem: A language is regular if and only if there exists some regular expression that describes it

Proof consists of two parts

1. If a language is described by a regular expression, then it is regular

2. If a language is regular, then it can be described by a regular expression

Claim 2. If a language is regular, then it can be described by a regular expression

Proof idea: We know: If L is a regular language then L is accepted by a deterministic finite automaton M

- We describe a procedure that converts M into a regular expression
- To do this we transform M into a *hybrid automaton* G—its definition is between automaton and regular expression(s)—that we then shrink until we **obtain the regular expression** that recognizes the same language as our original automaton does

Preparation

- Given **DFA** $M = (Q, \Sigma, \delta, q_0, F)$

- Create new generalized (nondeterministic finite) automaton

Generalized automaton G

- G has exactly one start state and is almost like an NFA, except:
  - Transitions in the state diagram are labeled with regular expressions
  - Exactly one accept state
  - Exactly one transition from every state to every other state (including same state; self loops)
    - Exceptions: no transition from accept state, no transition to the start state

Preparation

- Given DFA $M = (Q, \Sigma, \delta, q_0, F)$
- Create new generalized (nondeterministic finite) automaton G

1. Add new start state s and ϵ-transition from s to q_0
2. G has one accept state
   - Add new accept state f
   - ϵ-transitions from all states in F to f
   - convert M's accept states into non-accept states for G
3. Transform label on each transition into regular expression:
   - e.g., if label is a, b change it into $a \cup b$

Note that
$L(G) = L(M)$

Note that language does not change, only terminology

Preparation (continued)

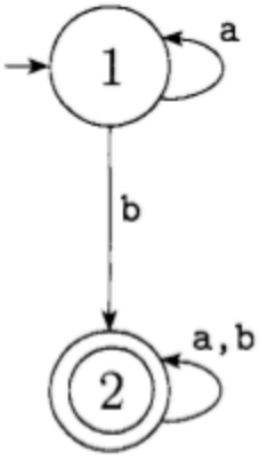
4. For each pair of states q_a and q_b with more than one transition from q_a to q_b, combine to one transition

5. For each pair of states q_a and q_b with no transition from q_a to q_b, add a transition and label it with $\emptyset$

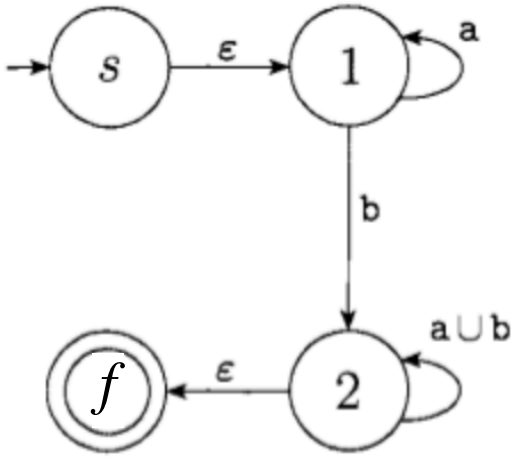
Note that language does not change, only terminology

Example

DFA



GNFA



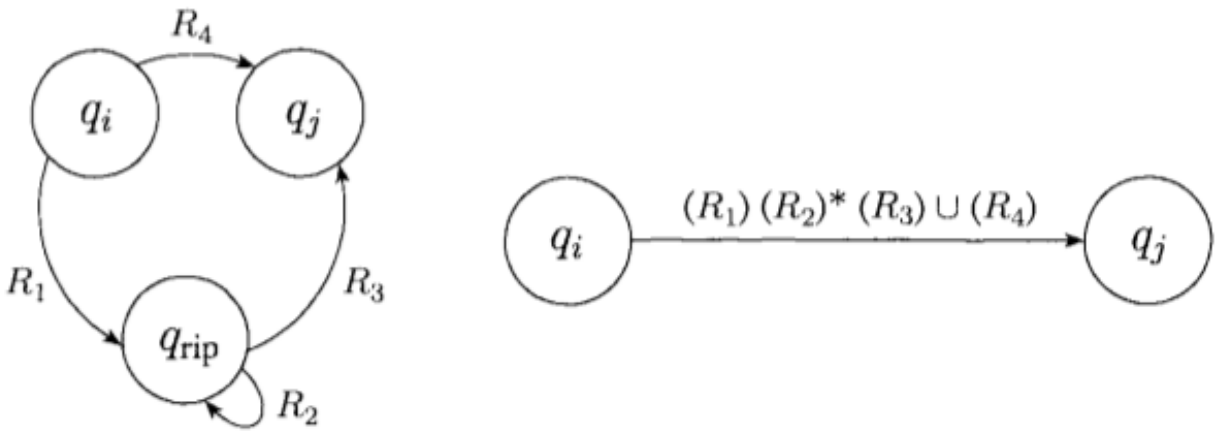
∅-transitions are omitted

From GNFA G to regex R for M with $L(M) = L(R)$

- We shrink G by removing states, until the only states left are s and f

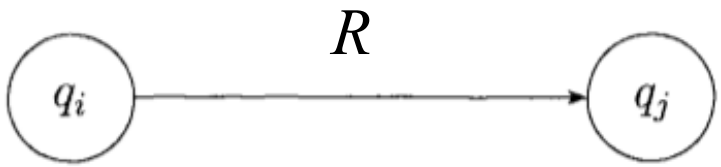
Transforming G into regular expression R

- Process to shrink/simplify G
  - remove states from Q (that is neither s or f), one by one, ensuring that G does not change language it recognizes
- Consider state to be removed: q_{rip}
- When removed, transitions through q_{rip} must be replaced



Shrinking

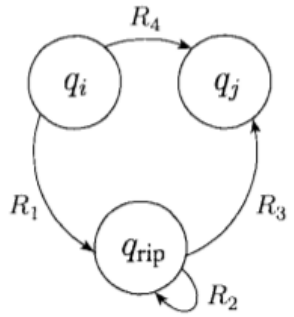
- Consider $G = (Q \cup \{s,f\}, \Sigma, \delta, s, f)$
- For states $q_i, q_j \in Q \cup \{s,f\}$ define reg$(q_i, q_j) = R$
  - R is regular expression with $\delta(q_i, R) = q_j$

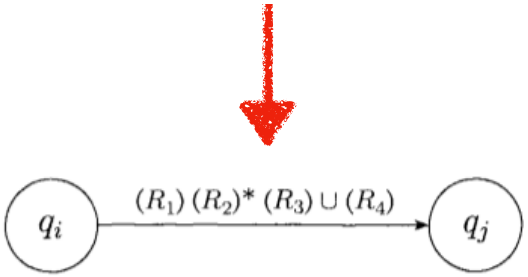


Shrinking (2)

Remove q_{rip}

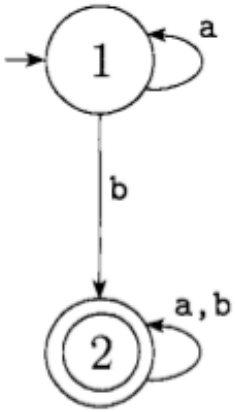
- Choose a state $q_{\text{rip}} \in Q$ and transform machine $G = (Q \cup \{s,f\}, \Sigma, \delta, s, f)$ to $G' = (Q', \Sigma, \delta', s, f)$ with $Q' = Q \cup \{s,f\} - \{q_{\text{rip}}\}$ and update δ to δ'
  - For each $q_i, q_j \in Q'$: $\text{reg}'(q_i, q_j) := (R_1)(R_2)^*(R_3) \cup (R_4)$ where
    - $R_1 = \text{reg}(q_i, q_{\text{rip}})$
    - $R_2 = \text{reg}(q_{\text{rip}}, q_{\text{rip}})$
    - $R_3 = \text{reg}(q_{\text{rip}}, q_j)$
    - $R_4 = \text{reg}(q_i, q_j)$



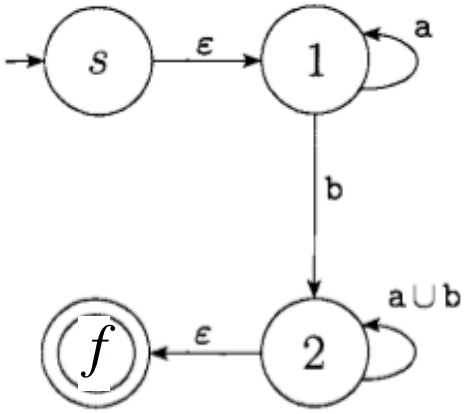


Process for ripping out one state

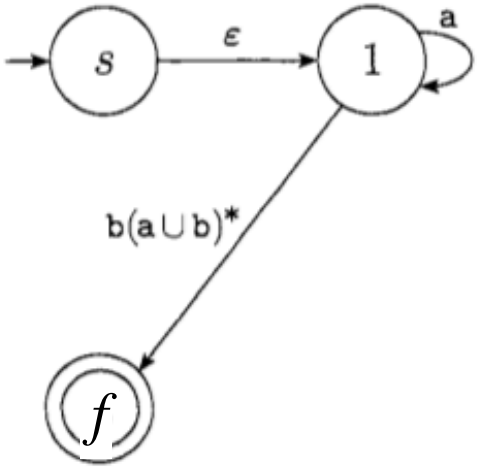
Example



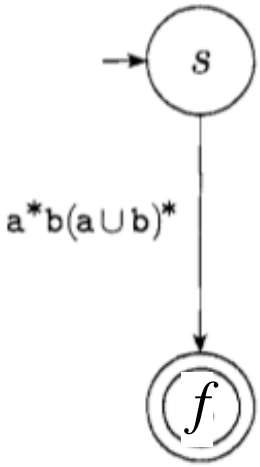
(a)



(b)

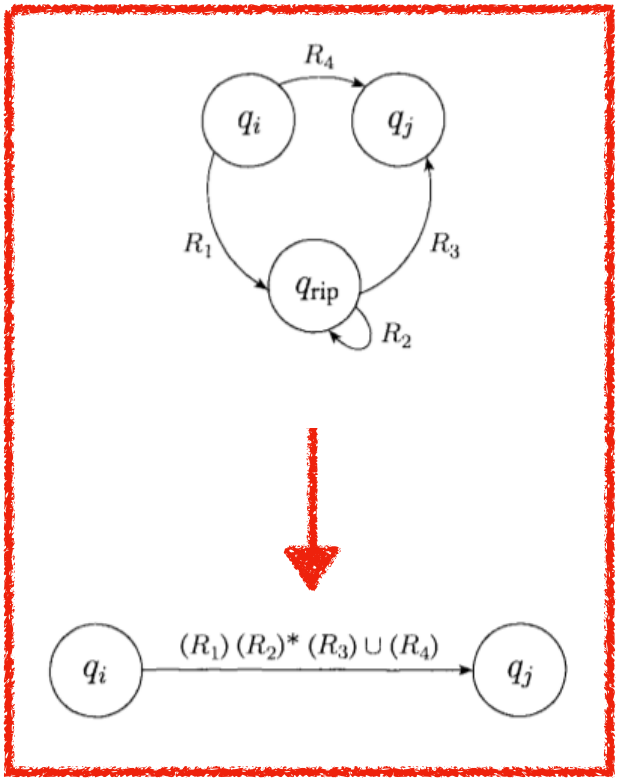


(c)



(d)

Transitions labeled with ∅ not shown

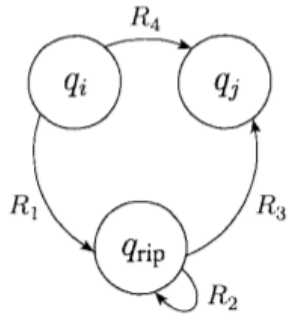


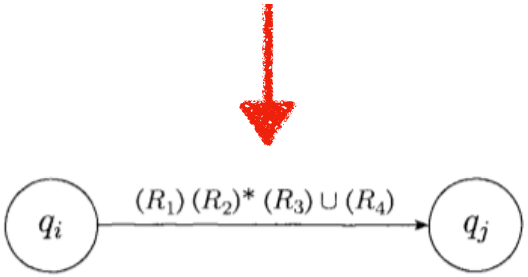
after

Shrinking (2)

Remove q_{rip}

- Choose a state $q_{\text{rip}} \in Q$ and transform machine $G = (Q \cup \{s,f\}, \Sigma, \delta, s, f)$ to $G' = (Q', \Sigma, \delta', s, f)$ with $Q' = Q \cup \{s,f\} - \{q_{\text{rip}}\}$ and update δ to δ'
  - For each $q_i, q_j \in Q'$: $\text{reg}'(q_i, q_j) := (R_1)(R_2)^*(R_3) \cup (R_4)$ where
    - $R_1 = \text{reg}(q_i, q_{\text{rip}})$
    - $R_2 = \text{reg}(q_{\text{rip}}, q_{\text{rip}})$
    - $R_3 = \text{reg}(q_{\text{rip}}, q_j)$
    - $R_4 = \text{reg}(q_i, q_j)$



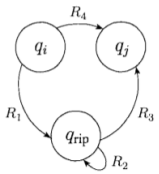


Process for ripping out one state

Finishing up …

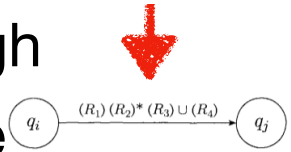
- The regular expression R obtained that labels the transition from s to f is the regular expression that describes $L(M)$
- Still to do: prove that $L(R) = L(M)$
  - we show that in every step of state removal $L(G) = L(G')$

We show: $L(G) = L(G')$



- $L(G) \subseteq L(G')$

1. When removing state q_{rip}: Every string accepted by G through transitions that **did not** pass through q_{rip} remains in language



2. Consider string accepted by G via transitions passing through q_{rip}

   - Removing q_{rip} from accepting sequence of states in G yields accepting sequence of states in G'

     - let …, q_i, q_{rip}, q_{rip},…, q_{rip}, q_j,... be sequence of states during computation in G

     - $reg'(q_i,q_j)$ includes any substring recognized through …,q_i,q_j,....

     - Thus ...,q_i,q_j,... is accepting computation in G'

We show: $L(G) = L(G')$

- $L(G') \subseteq L(G)$
  - G' accepts string w
  - w accepted by G' corresponds to the concatenation of labels on path in G
  - w must have been accepted by G

Theorem: A language is regular if and only if there exists some regular expression that describes it

Summary

1. If a language is described by a regular expression, then it is regular

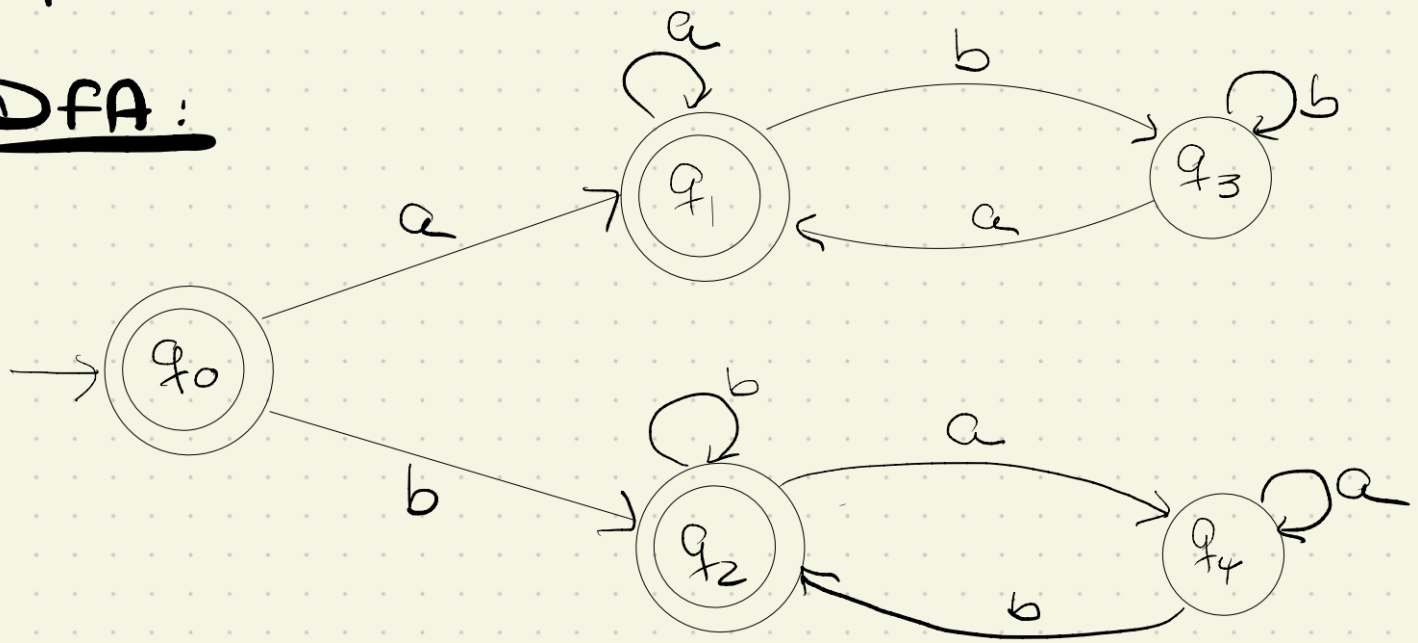
Regular expression ⟶ NFA ⟶ regular language

2. If a language is regular, then it can be described by a regular expression

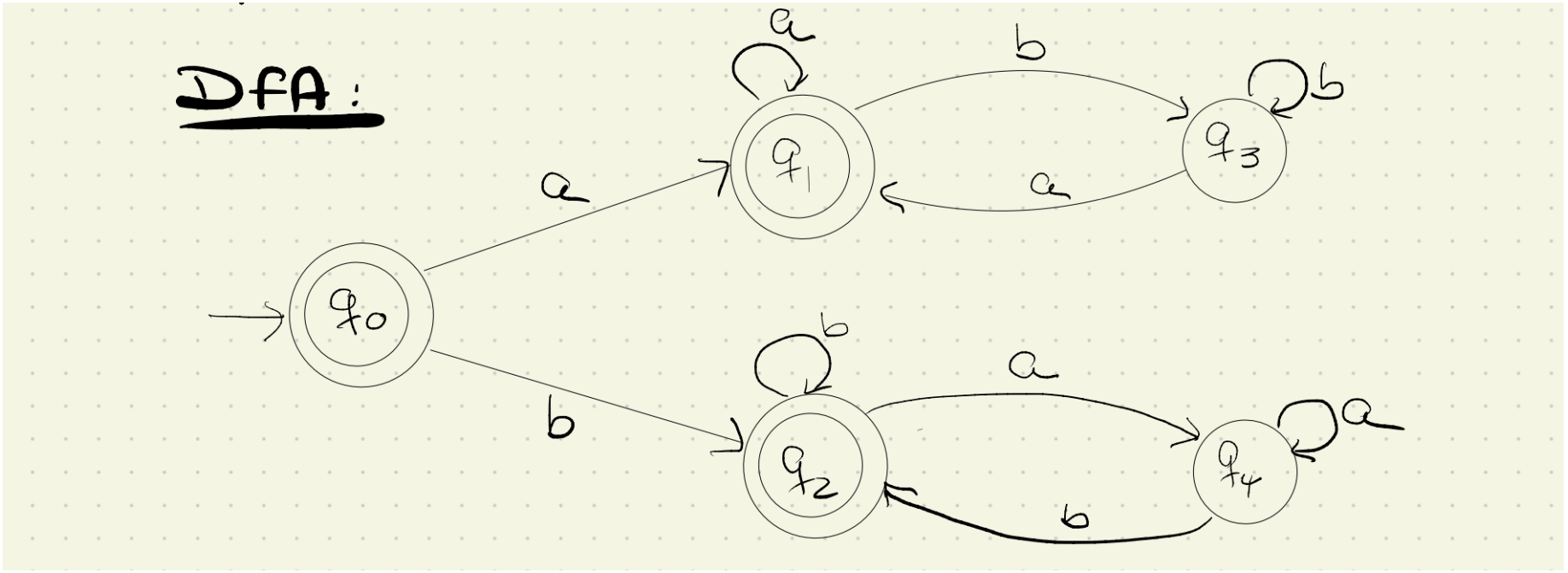
Regular language ⟶ DFA ⟶ GNFA⟶ regular expression

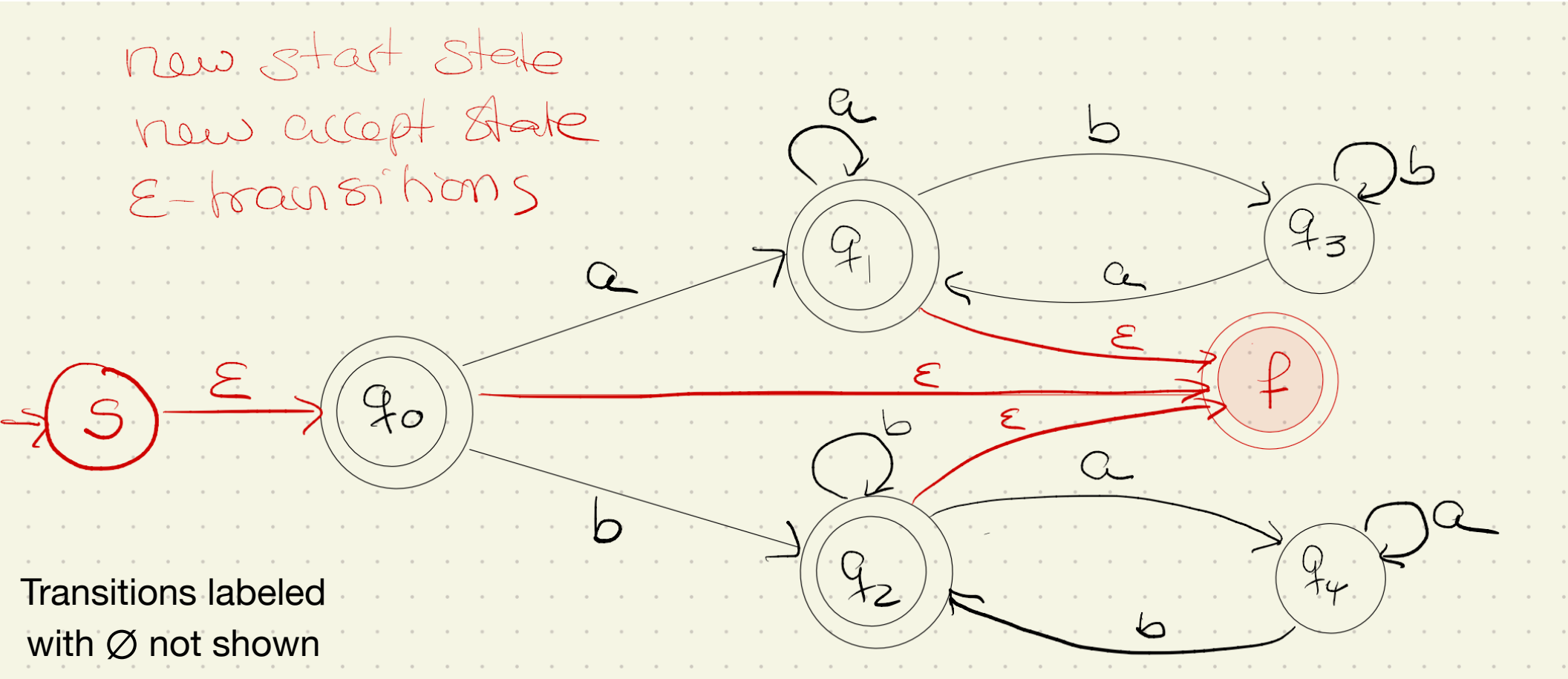
Example : DFA to Regular Expression

DFA:



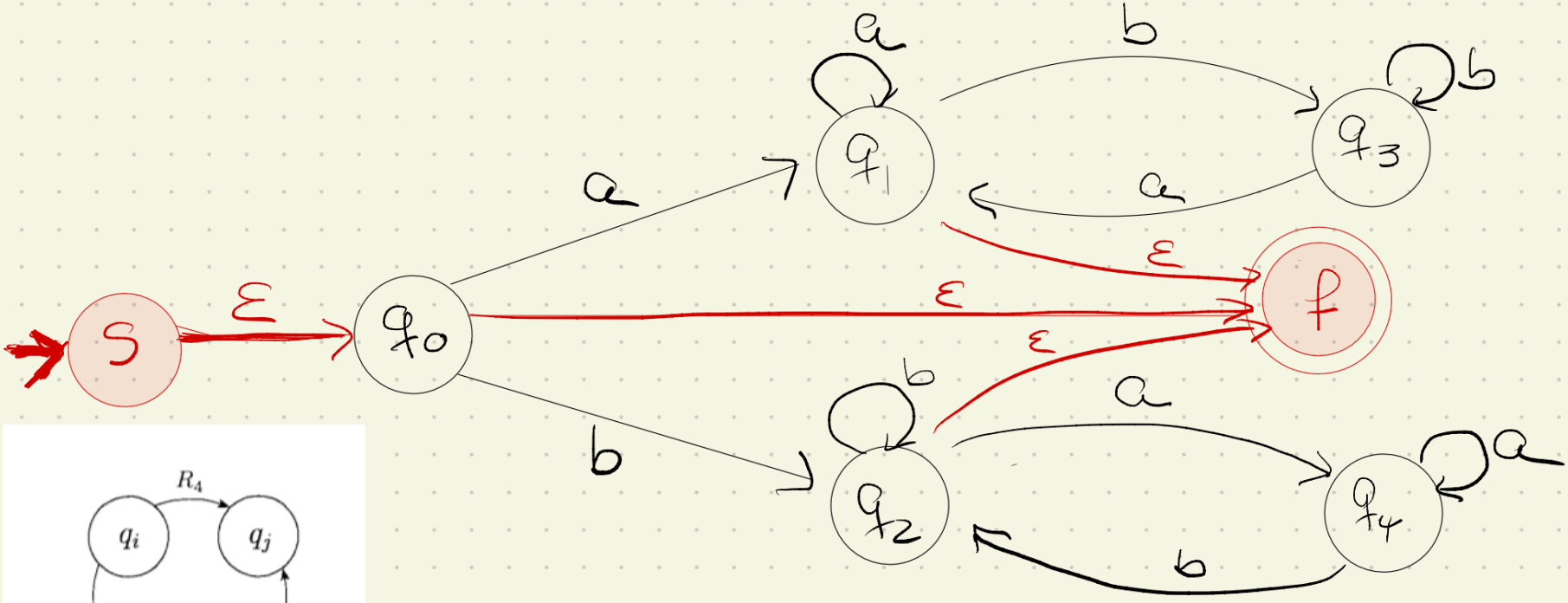
Create generalized automaton

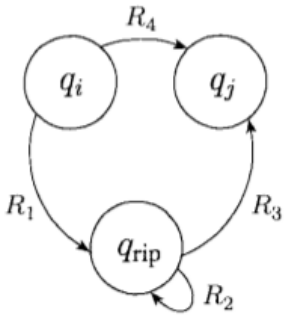

DFA:
a
b
a
b
a
q0
q1
q3
b
a
b
a
q2
q4
b

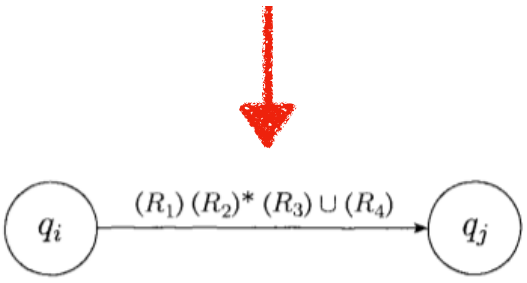

new start State
new accept State
ε-transitions
a
b
a
b
a
q1
q3
ε
ε
ε
S
ε
q0
f
b
a
b
q2
q4
b
Transitions labeled
with ∅ not shown

q_0 is no start state anylonger

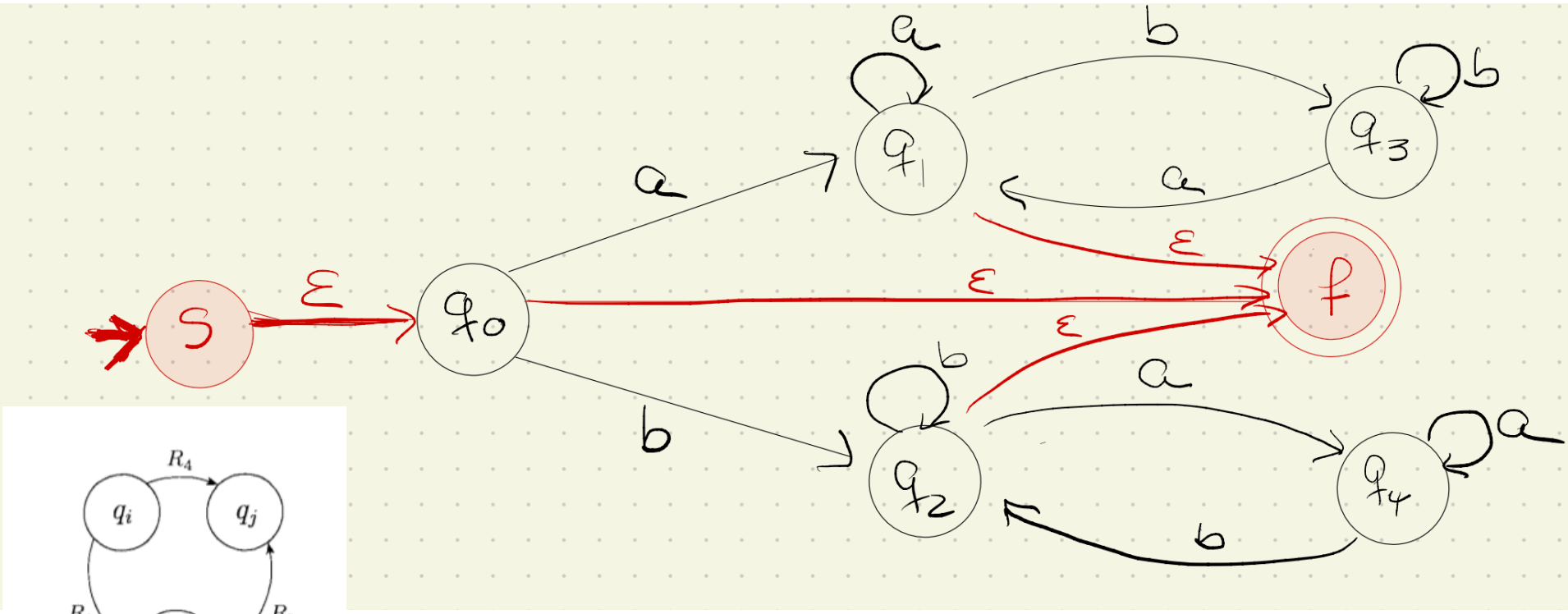
f is now the only accept state

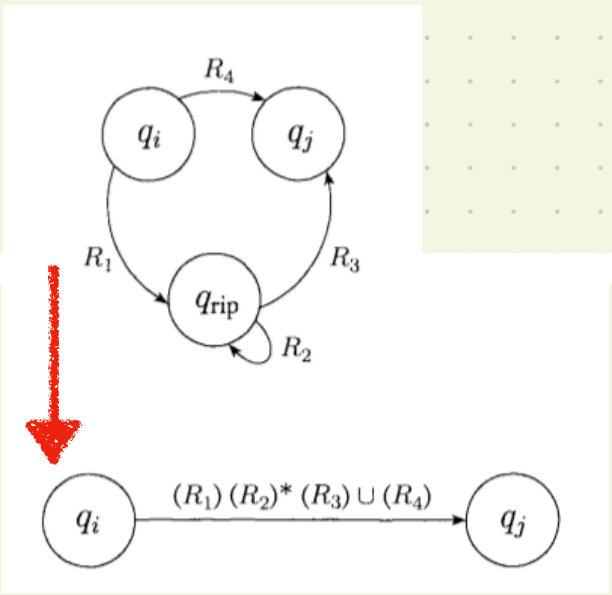


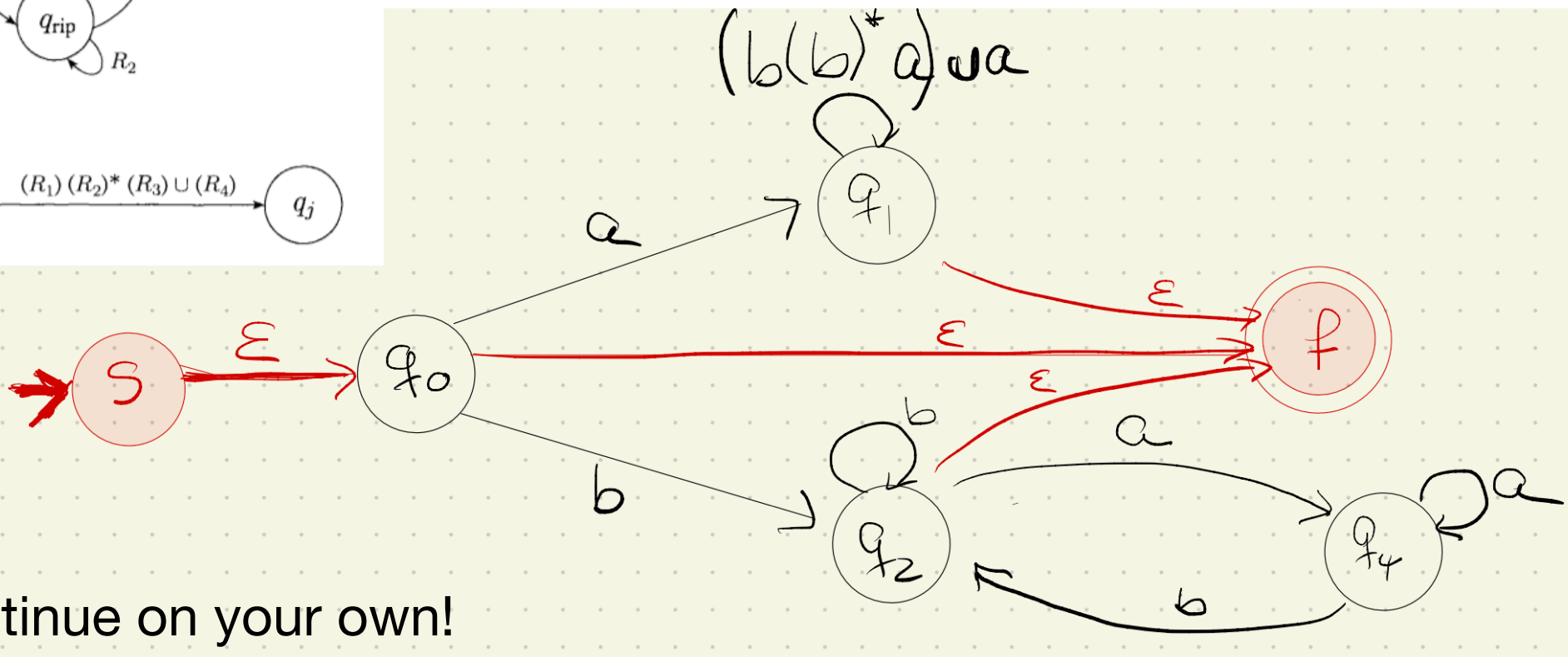




Your turn: Remove q_3







Continue on your own!

Next: State Minimization of DFAs

DFA State Minimization

- Given a DFA
- Goal: reduce number of states without changing language recognized
    1. Remove unreachable states
    2. Identify/collapse states that yield the same result
        - Maintain determinism

What kind of states can/ cannot be collapsed?

- Don’t collapse accept and non-accept states
  - If there exists a string w and states p, q such that
    - when w is processed starting at state p M yields acceptance and
    - when w is processed starting at state q M yields non-acceptance
    - Then do not collapse p and q

State equivalence

- Two states p, q of DFA M are **equivalent** (**$p \sim q$**) if and only if *for all* $w \in \Sigma^*$
  - computation of M for w starting at state p ends in accept state if and only if computation of M for w starting at state q ends in accept state