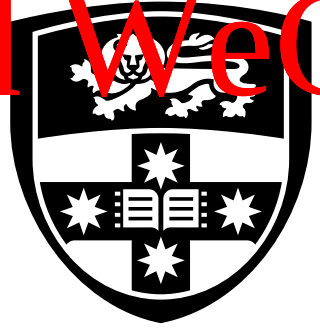
COMP2022: Formal Languages and Logic

2018, Semester 2, Week 6



THE UNIVERSITY OF SYDNEY

Outline

- **Regular Expressions**
-
-
- Proving if a language is, *or is not*

Outline

- Regular Expressions
-
-
- Proving if a language is, *or is not*

Equivalence of RegEx and FA

Theorem:
A language is regular if and only if a regular expression describes it.

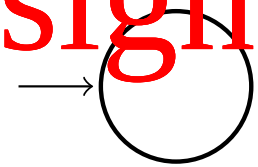
Proo
Sho

1. ⇒
   Show that for each RegEx, there exists an NF
   recognises the same language

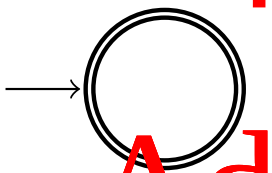
2. FA ⇒ RegEx:
   Show that for each NFA, there exists a RegEx which
   recognises the same language

From RegEx to FA: atomic cases

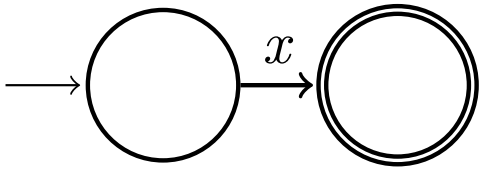
Automaton for $\emptyset$



Auto



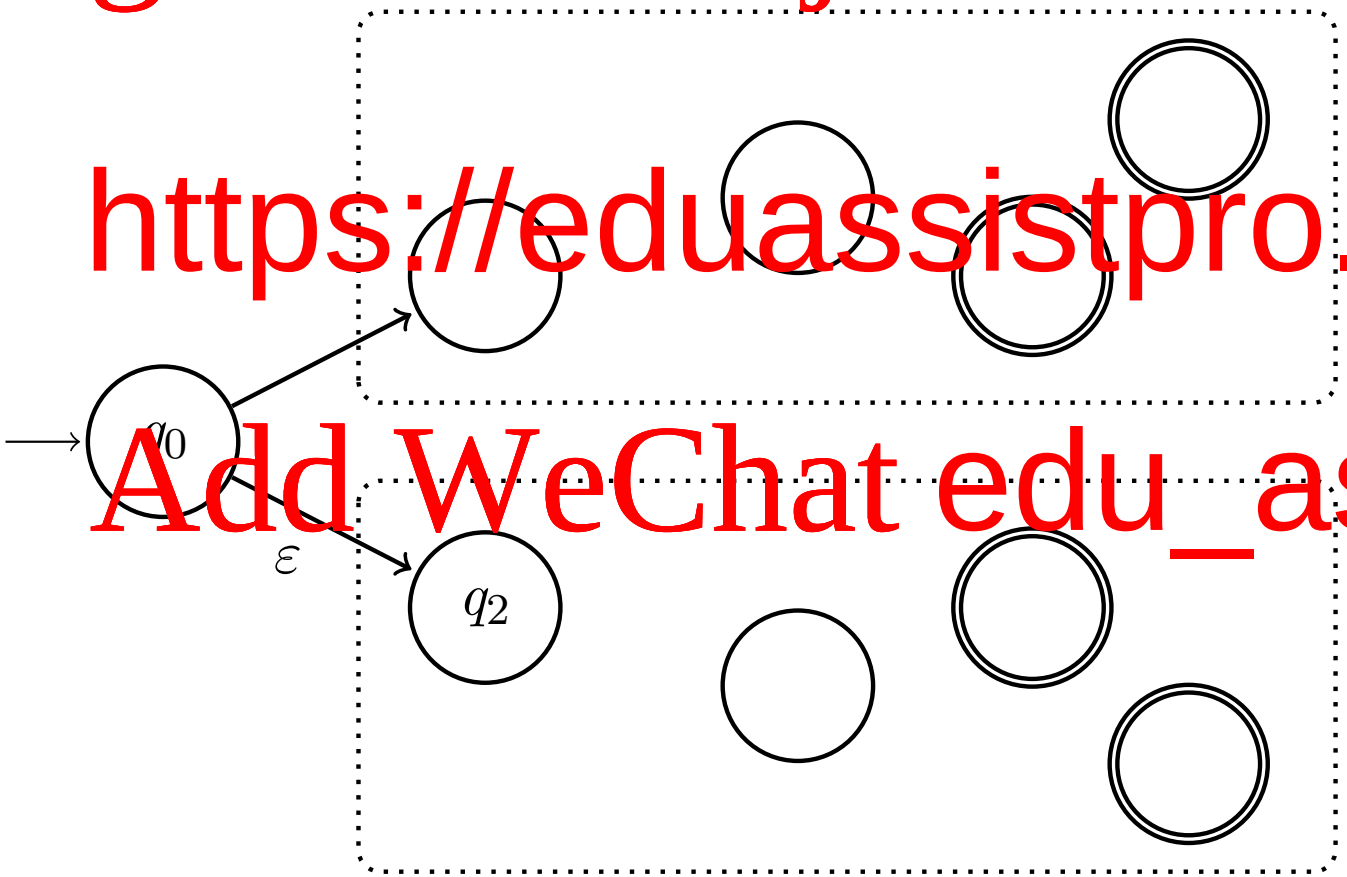
Automaton for $x \in \Sigma$



NFA for regular operations: Union

Let M_1 and M_2 be automata recognising $L(R_1)$ and $L(R_2)$

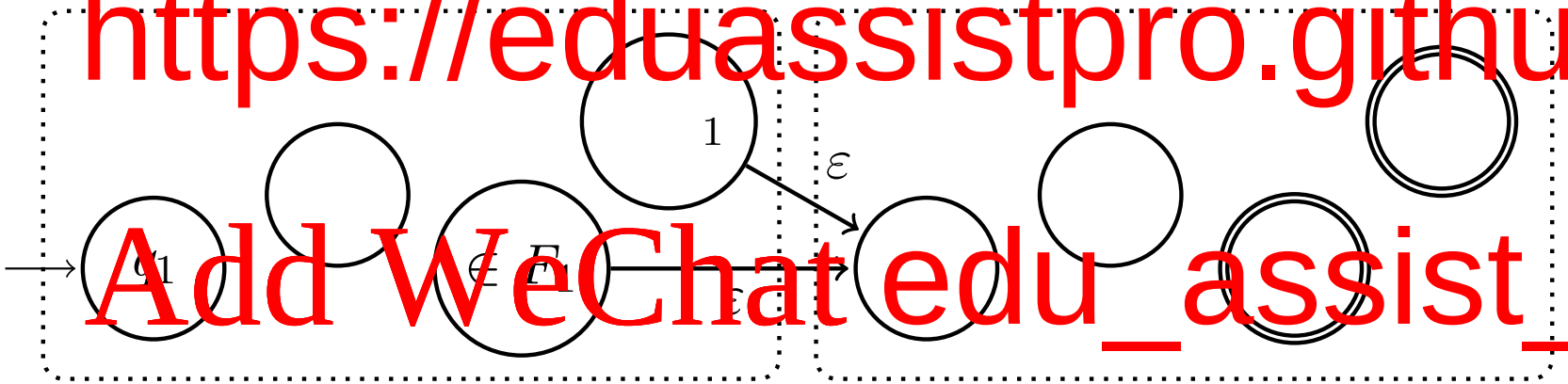
Then an automaton M for $R_1 \mid R_2$ is:



NFA for regular operations: Concatenation

Let M_1 and M_2 be automata recognising $L(R_1)$ and $L(R_2)$

Then an automaton M for $R \quad R$ is:

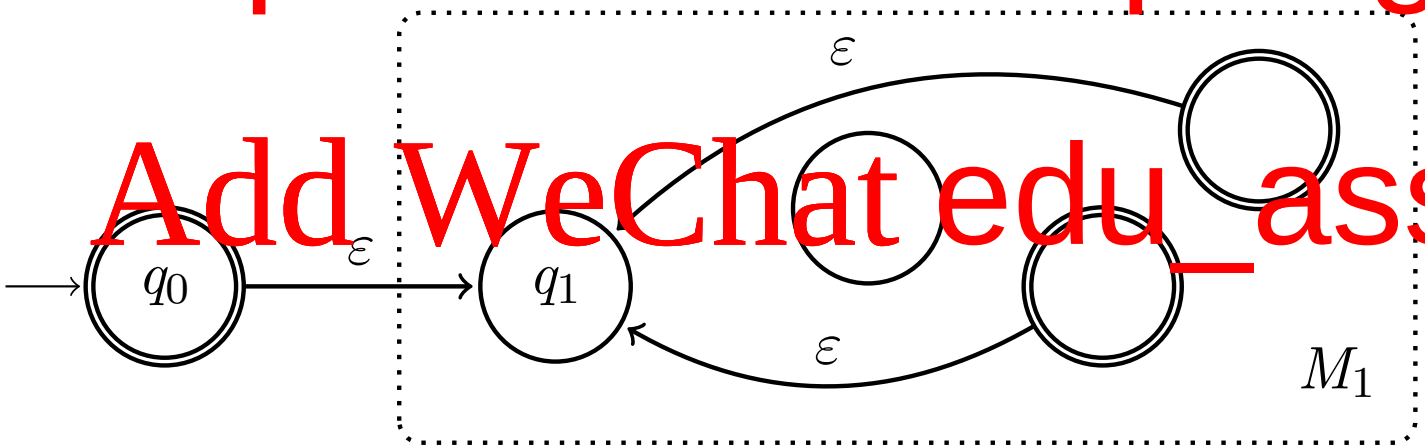


Reminder: the accept states of M_1 are *not* accept states in M

NFA for regular operations: Star closure

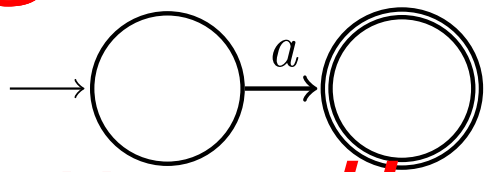
Let M_1 be an automaton recognising $L(R_1)$

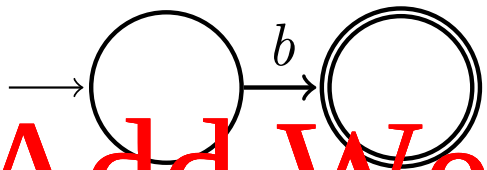
The

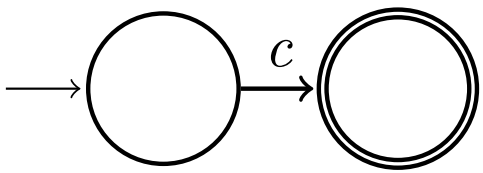


Example: $a \mid bc^\star$: from RegEx to NFA

1. Construct automata for the atomic regular expressions a, b, c

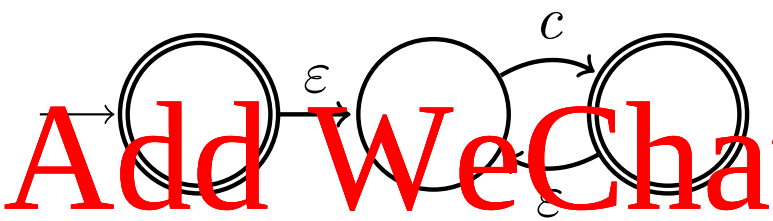






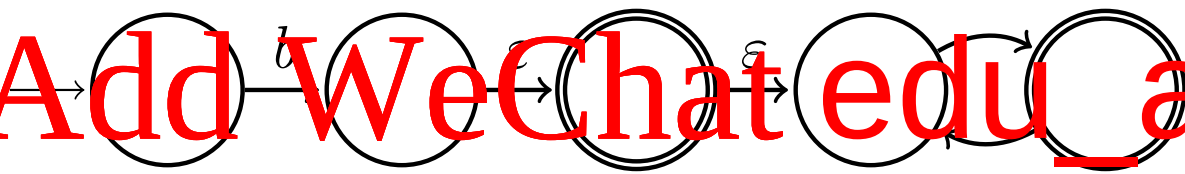
Example: $a \mid bc^\star$: from RegEx to NFA

1. Construct automata for the atomic regular expressions a, b, c
2. Use the Star Closure operation to find an automaton for $c^\star$



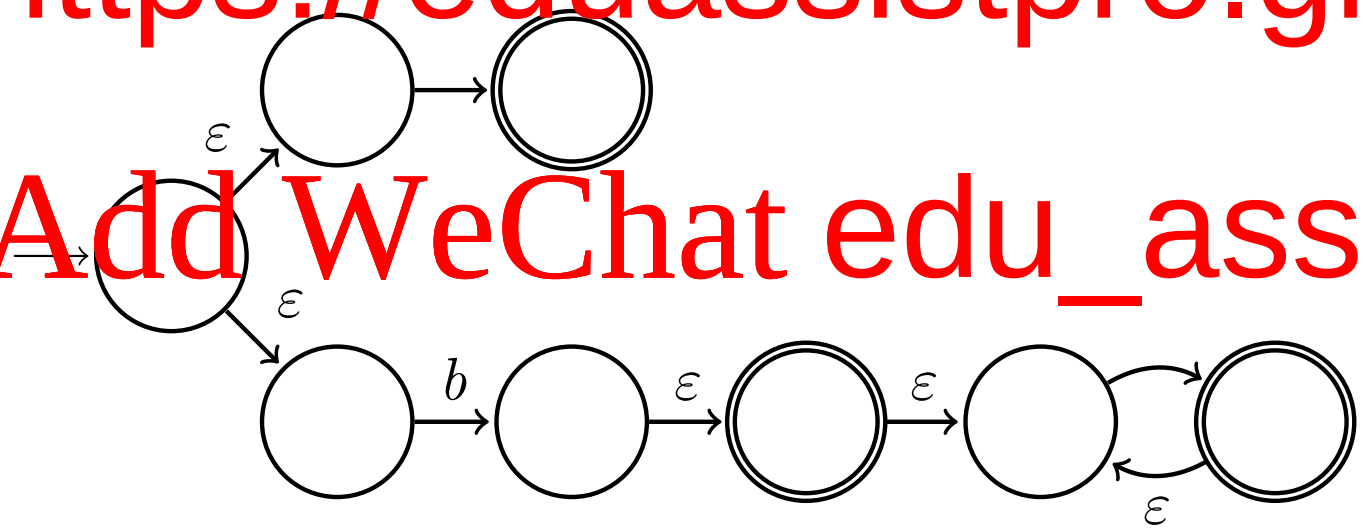
Example: $a \mid bc^\star$: from RegEx to NFA

1. Construct automata for the atomic regular expressions a, b, c
2. Use the Star Closure operation to find an automaton for $c^\star$
3. Use the Concatenation operation to find an automaton for $bc^\star$



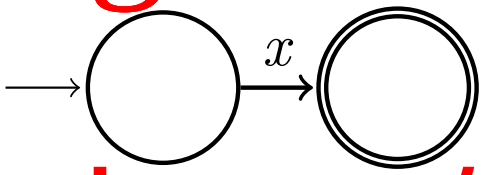
Example: $a \mid bc^\star$: from RegEx to NFA

1. Construct automata for the atomic regular expressions a, b, c
2. Use the Star Closure operation to find an automaton for $c^\star$
3. Use the Concatenation operation to find an automaton for $bc^\star$
4. $c^\star$

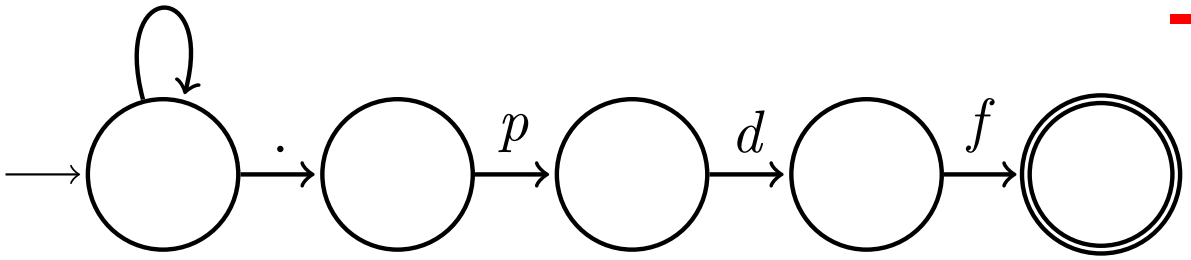


From NFA to RegEx: Simple examples

This automaton gives the RegEx x

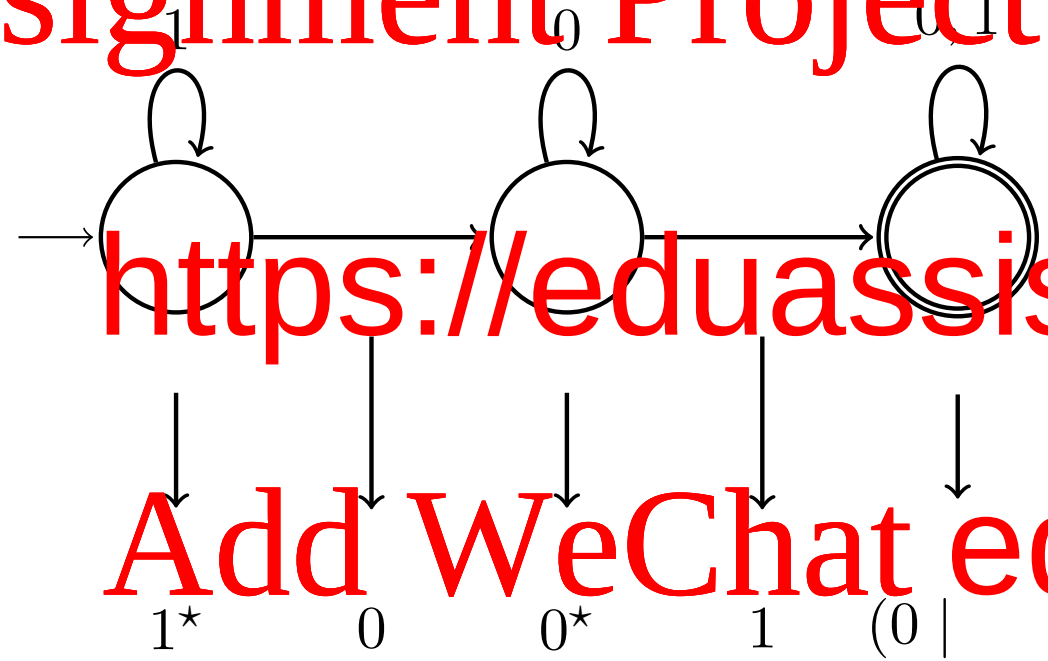


This o



From NFA to RegEx: Simple examples

A more complex one:



$1^\star \quad 0 \quad 0^\star \quad 1 \quad (0 \mid$

$1^\star 00^\star 1(0 \mid 1)^\star$

Concept

Algorithm to convert an NFA to a RegEx:

1. Convert the NFA to a Generalised NFA (GNFA)

2. Progressively *eliminate* all states bet state

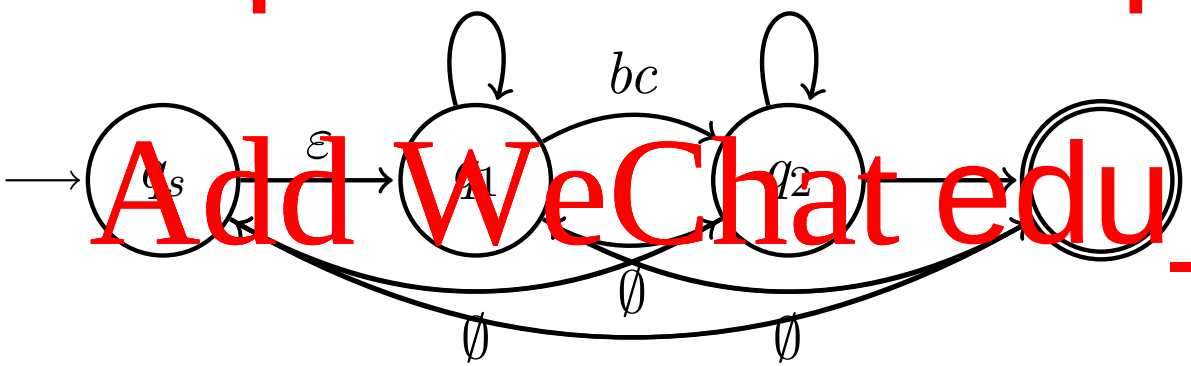
3. The transition between the start and the accept state is now a regular expression describing L

Generalised NFA (GNFA)

The start state q_s is non-accepting and has no incoming transitions

The only accept state q_a has no outgoing transitions, and $q_s \neq q_a$

There is exactly one transition between each ordered pair of states, label



$\{abc, abca, abcbb, aaaaaabc\} \subseteq L$

Generalised NFA (GNFA)

The start state q_s is non-accepting and has no incoming transitions

The only accept state q_a has no outgoing transitions, and $q_s \neq q_a$

There is exactly one transition between each ordered pair of states, label

We do not show the $\emptyset$ transitions (why not?)

$\{abc, abca, abcbb, aaaaaabc\} \subseteq L$

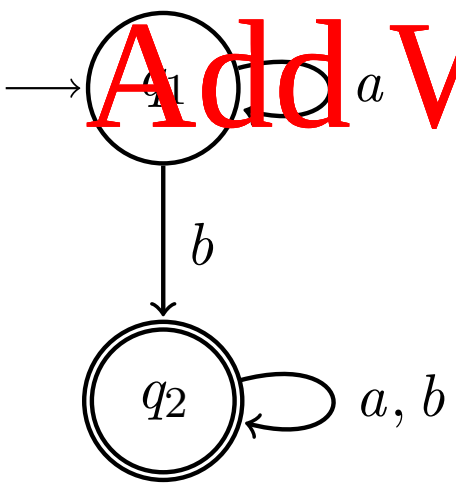
Converting an NFA to a GNFA

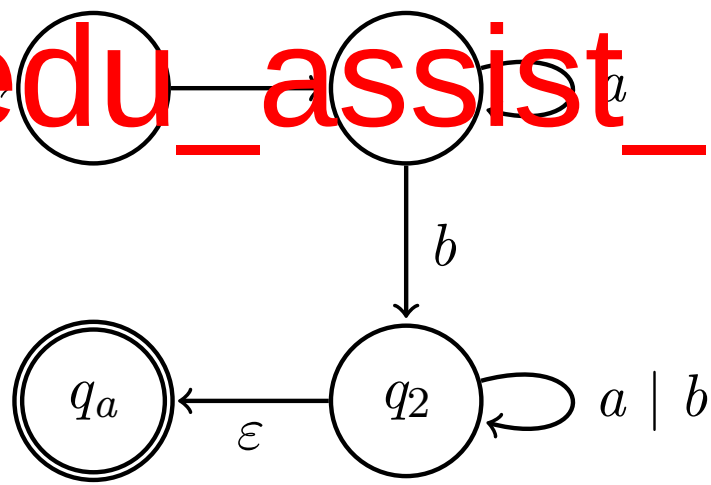
Create a new non-accepting start state q_s, with a ε-transition to the original start state.

Create a new accept state q_a with ε-transitions from the original acce

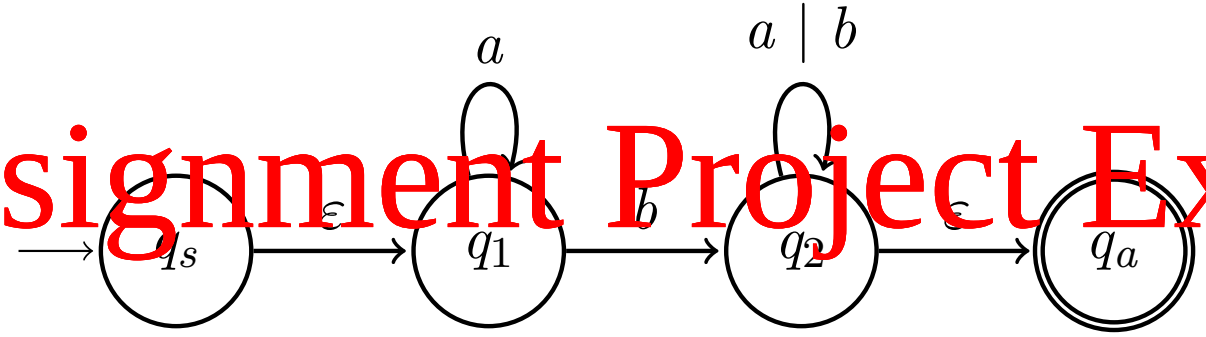
Labe tates from t trans

q_i q_j





Simple example: eliminate state q_1



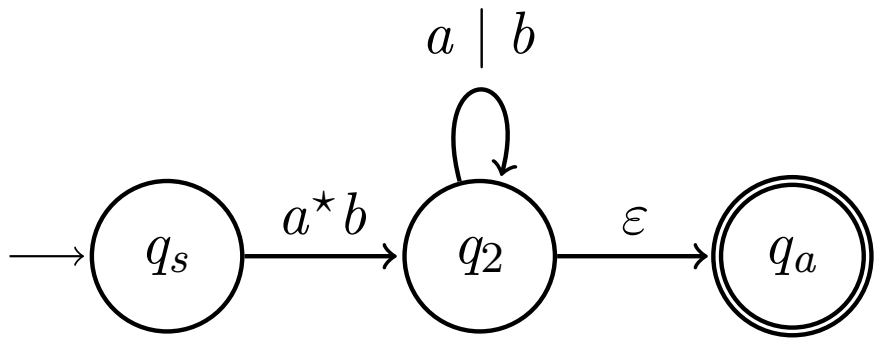
Ther

$q_1 \to q_1$ $a^\star$

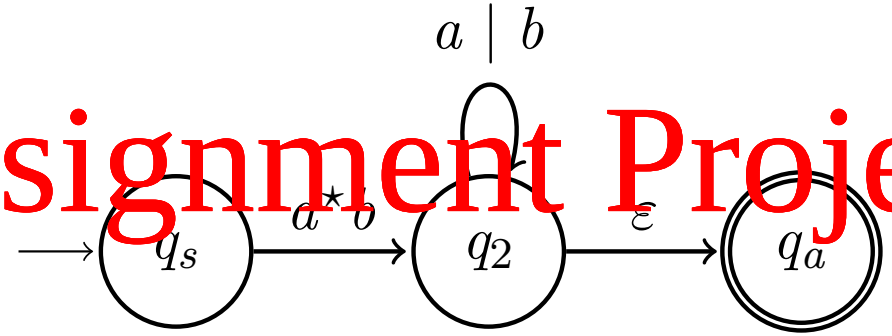
$q_1 \to q_2$

We set the transition $q_s \to q_2$ to the unio

it's old value ($\emptyset$). This is $\varepsilon a^\star b \mid \emptyset$, whic $a \;\; b$



Simple example: eliminate state q_2

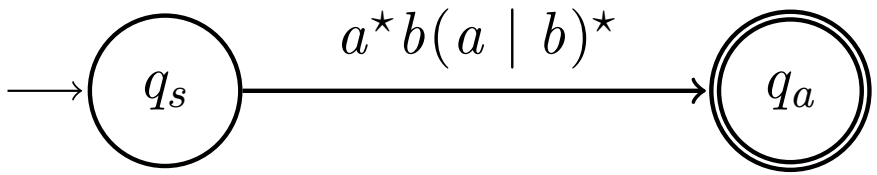


Ther

s 2

$q_2 \to q_2$ any number of times

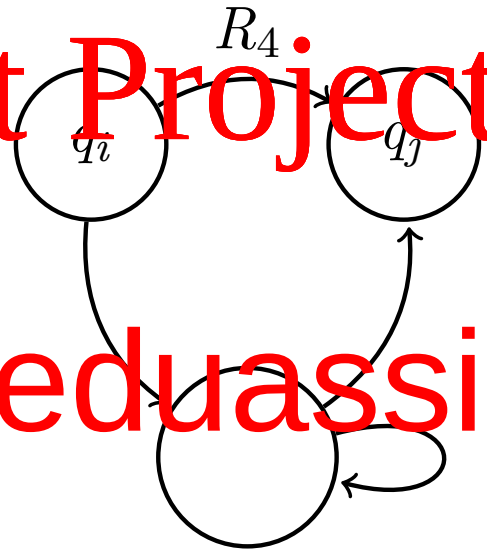
$q_s \to q_a$



The language of the original automaton is $a^\star b(a \mid b)^\star$

General method to eliminate state q_{elim}

Consider each pair (q_i, q_j) where $q_i, q_j \in Q \setminus \{q_{elim}\}$



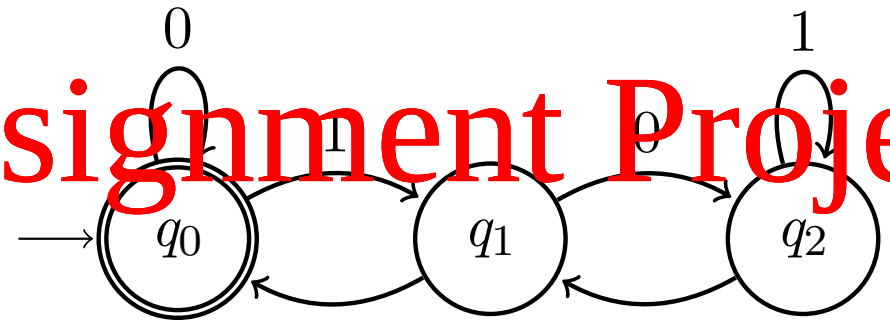
Replace the transition from q_i to q_j wi

$$((R_1)(R_2)^\star(R_3)$$

Note:

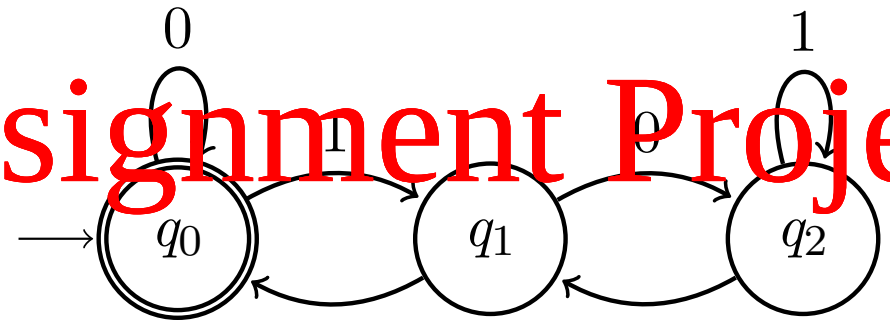
- Possibly $q_i = q_j$
- Recall that pairs are ordered, so we also consider (q_j, q_i)
- If there is no transition R_x, then $R_x = \emptyset$

Example 2: divisible by 3

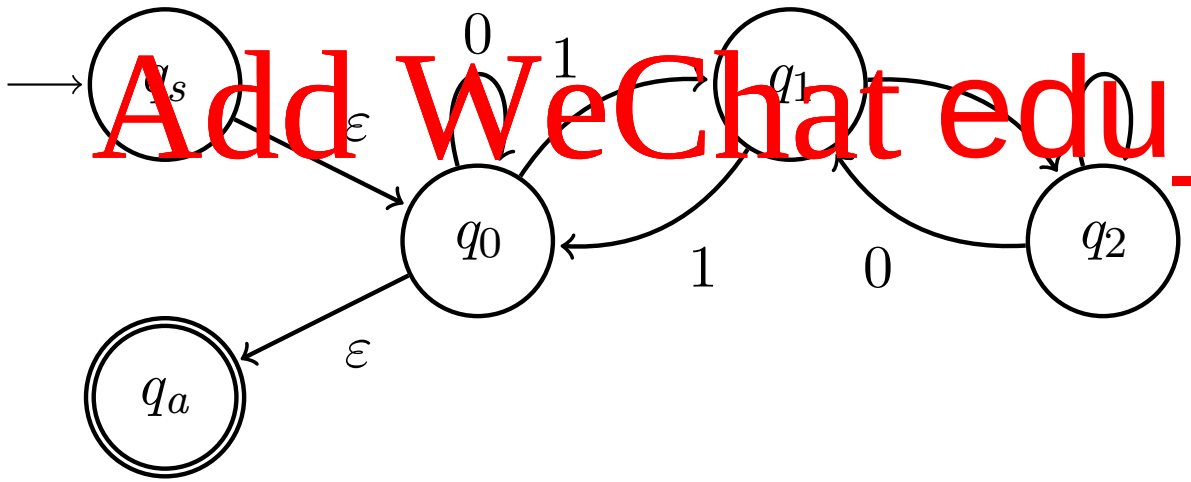


Conv

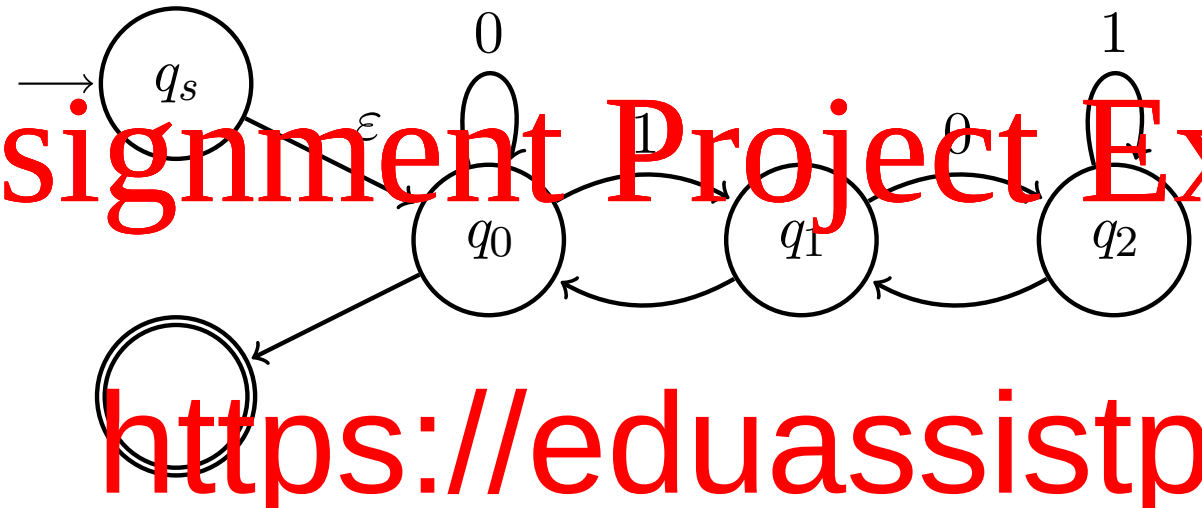
Example 2: divisible by 3



Conv



Example 2: divisible by 3

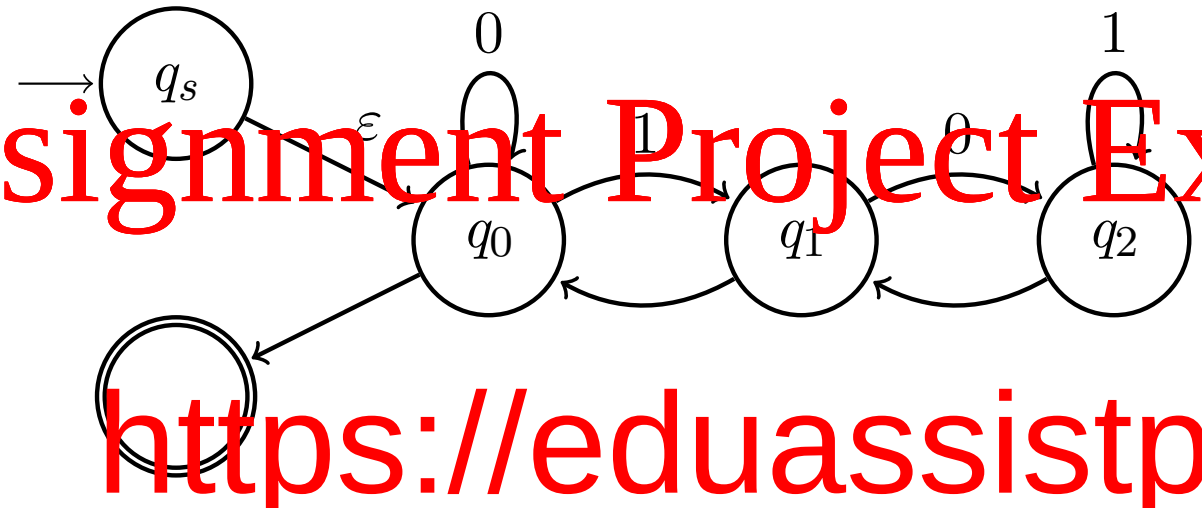


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- ▶
- ▶
- ▶
- ▶

Example 2: divisible by 3

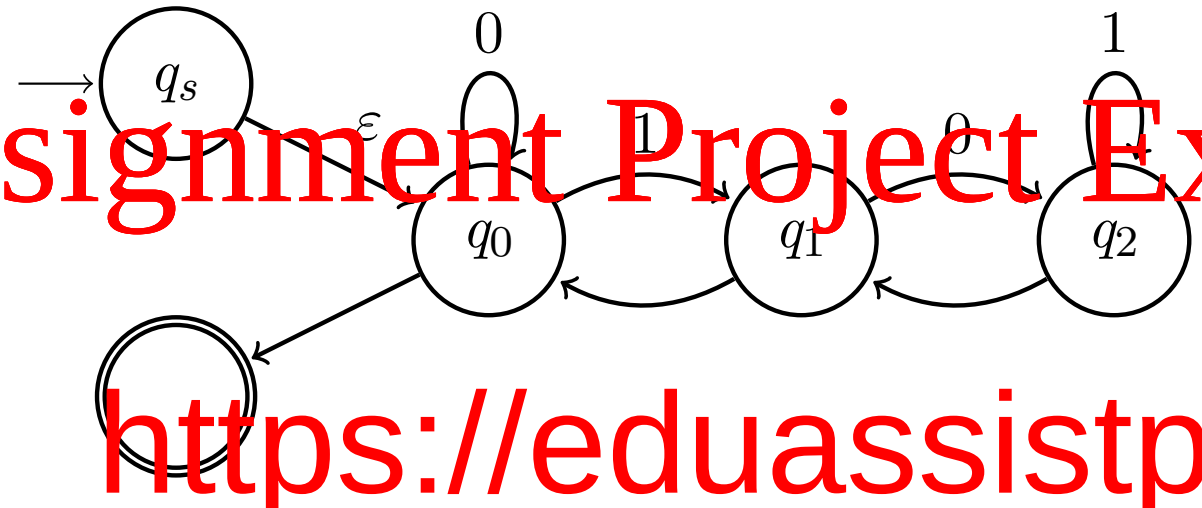


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- (q_s, q_1) :
- (q_s, q_a) :
- (q_1, q_1) :
- (q_1, q_a) :

Example 2: divisible by 3

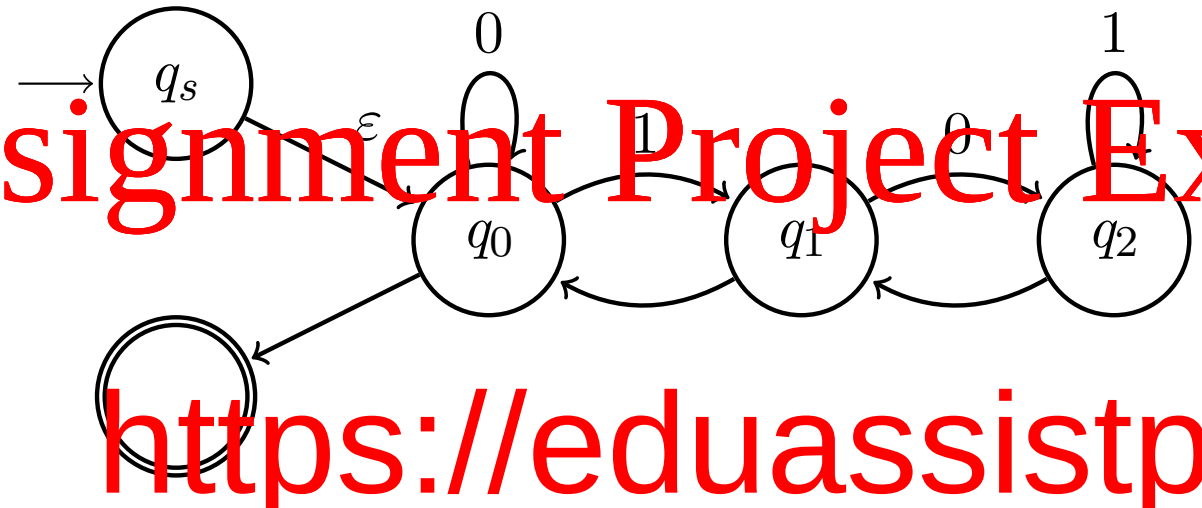


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon,$
- $(q_s, q_a) :$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

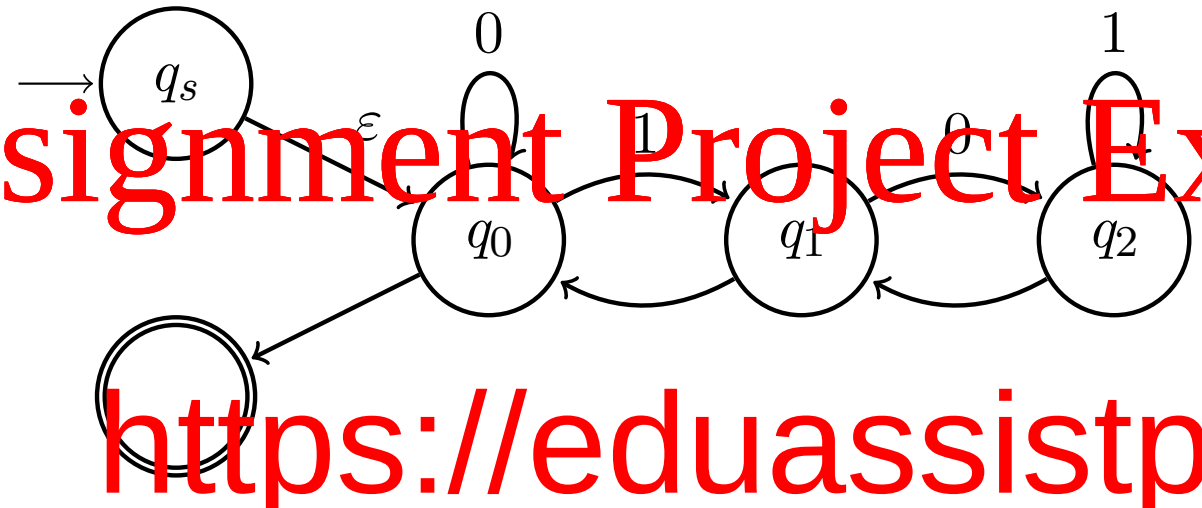


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0,$
- $(q_s, q_a) :$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

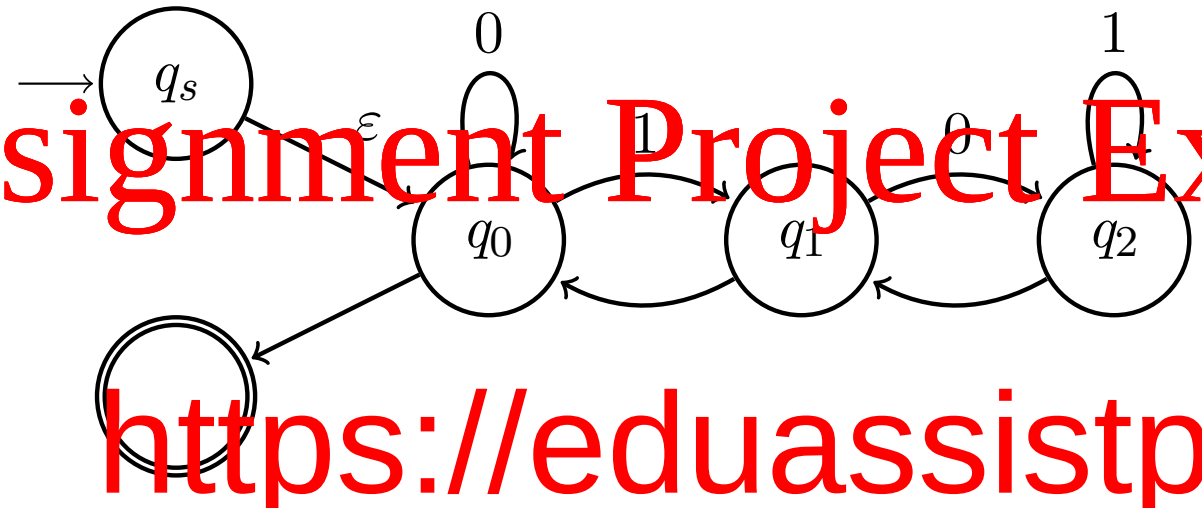


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) :$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

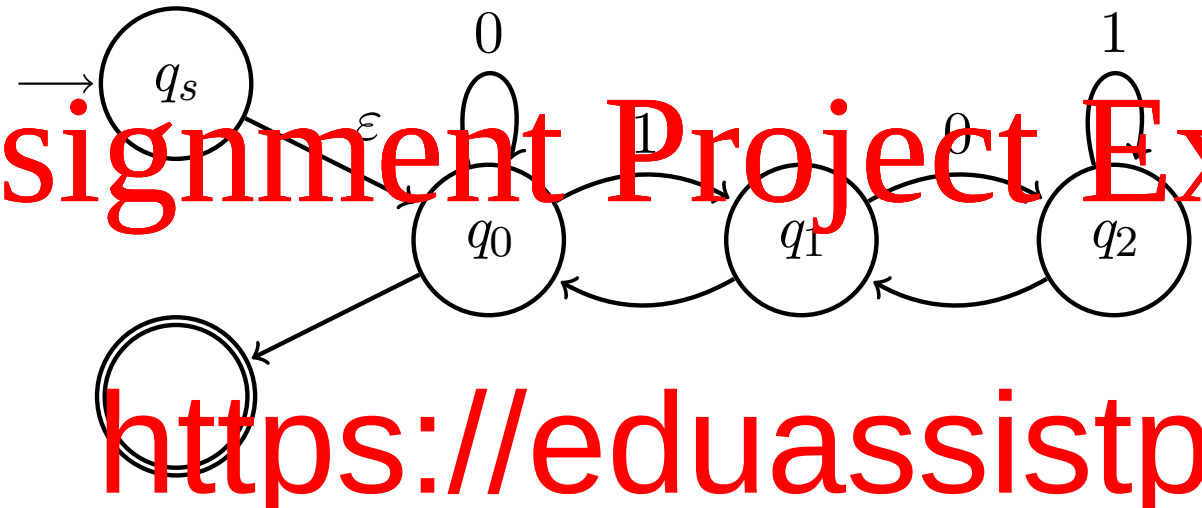


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) :$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

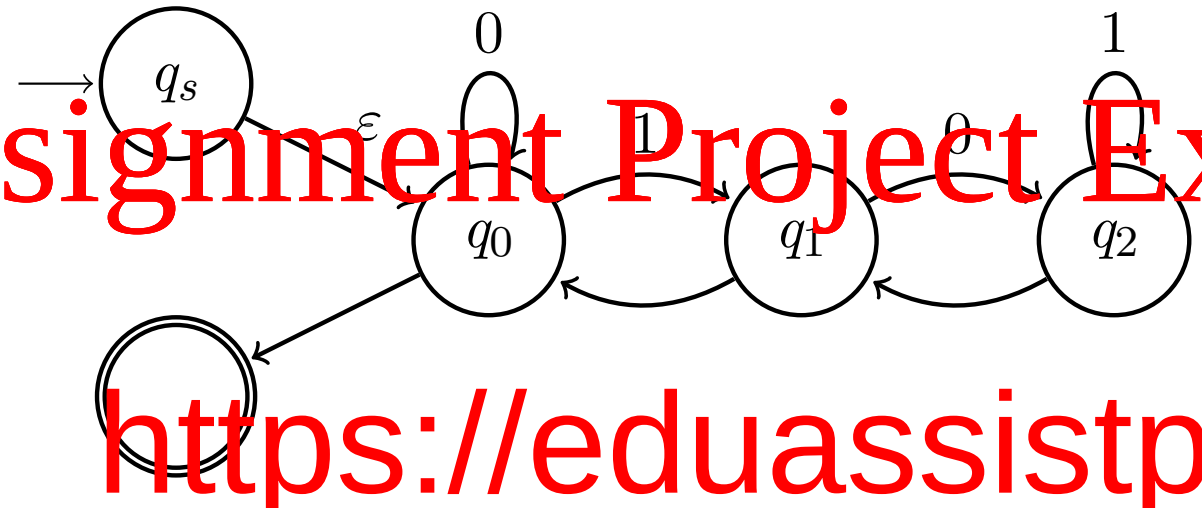


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) :$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

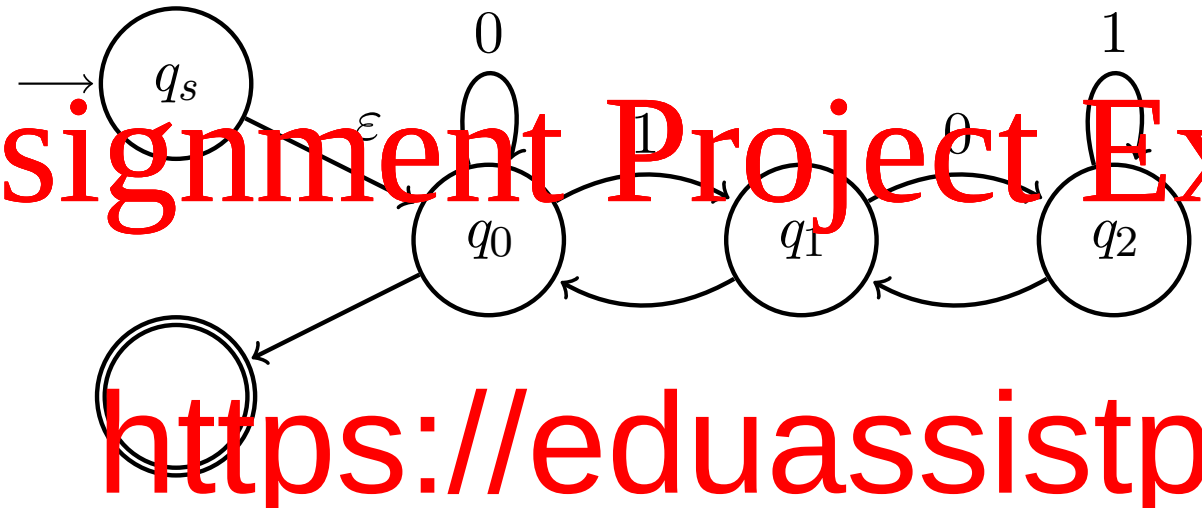


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) :$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

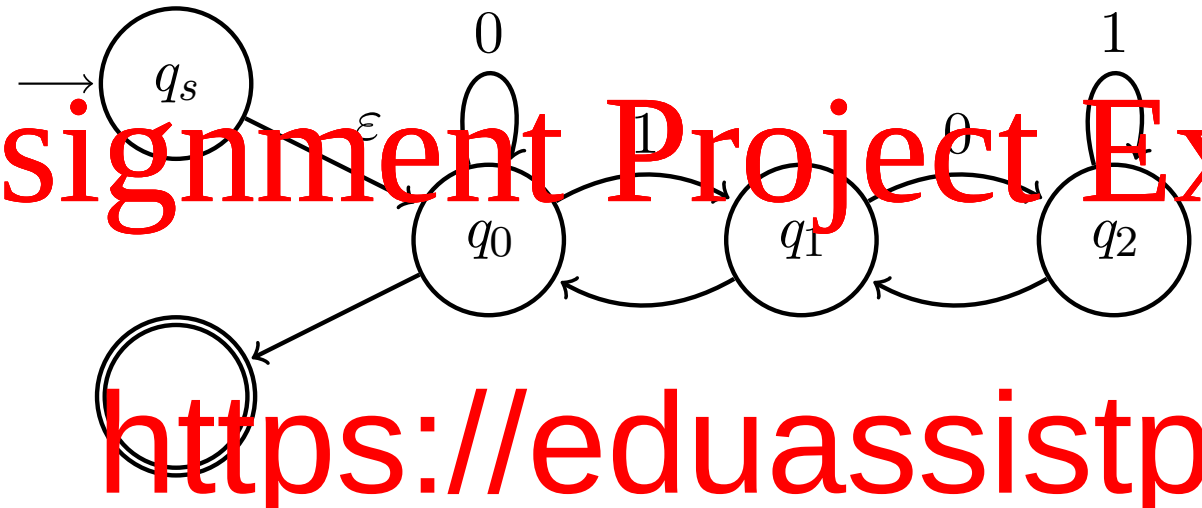


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) : R_1 = \varepsilon,$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

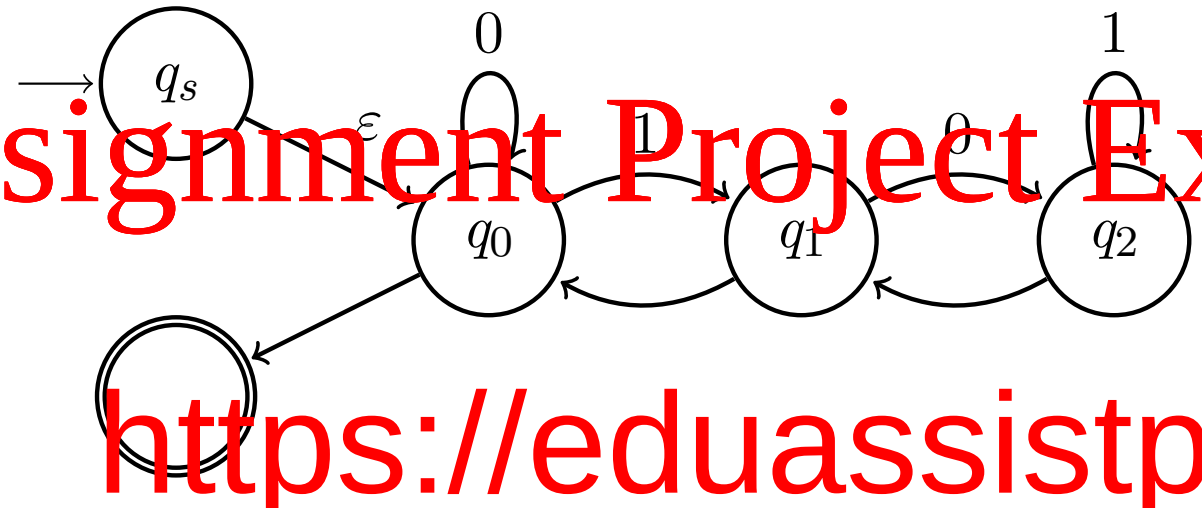


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) : R_1 = \varepsilon, R_2 = 0,$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

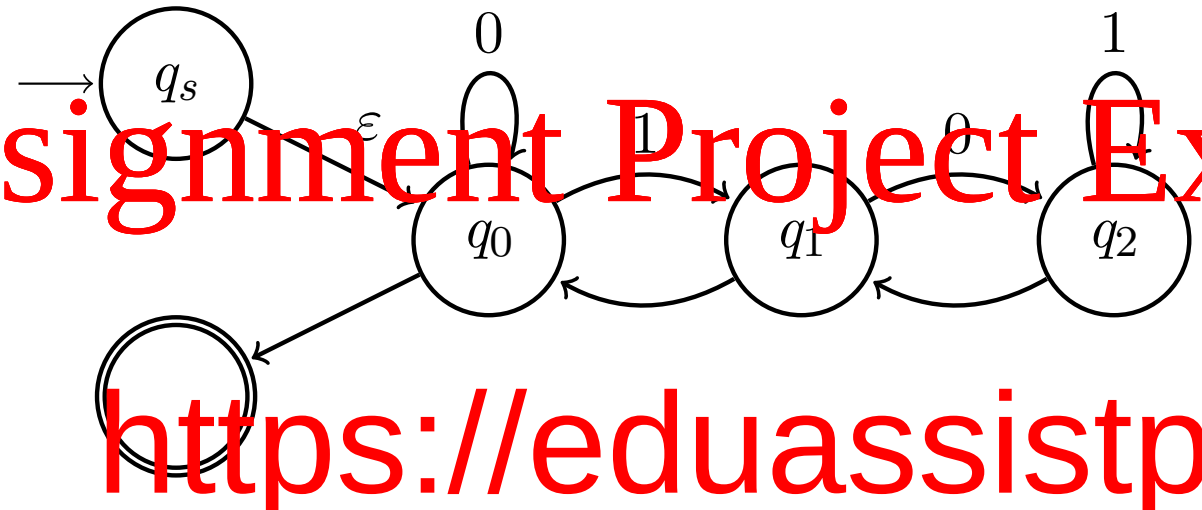


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) : R_1 = \varepsilon, R_2 = 0, R_3 = \varepsilon,$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

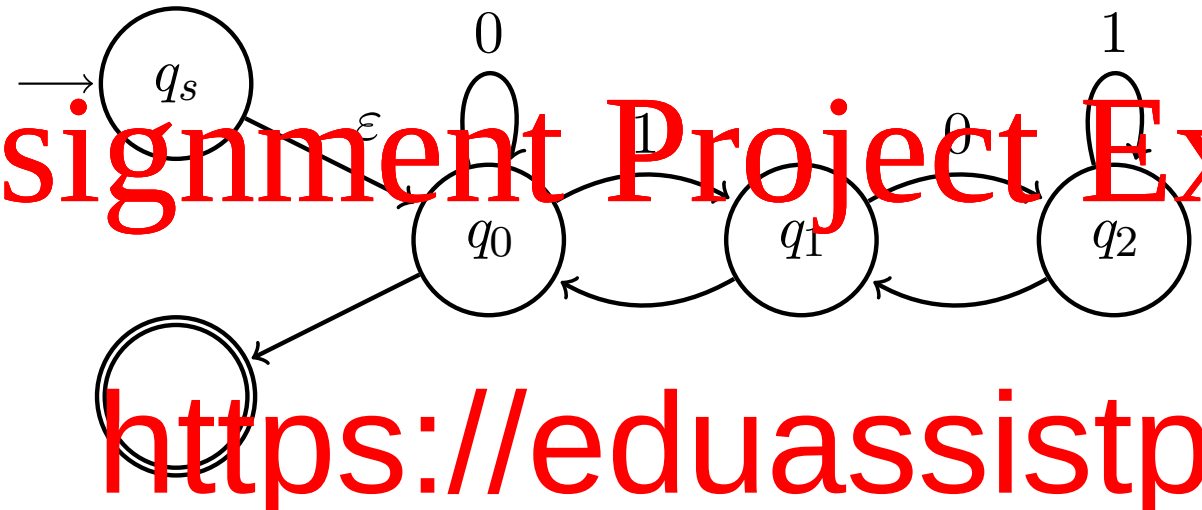


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) : R_1 = \varepsilon, R_2 = 0, R_3 = \varepsilon, R_4 = \emptyset$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

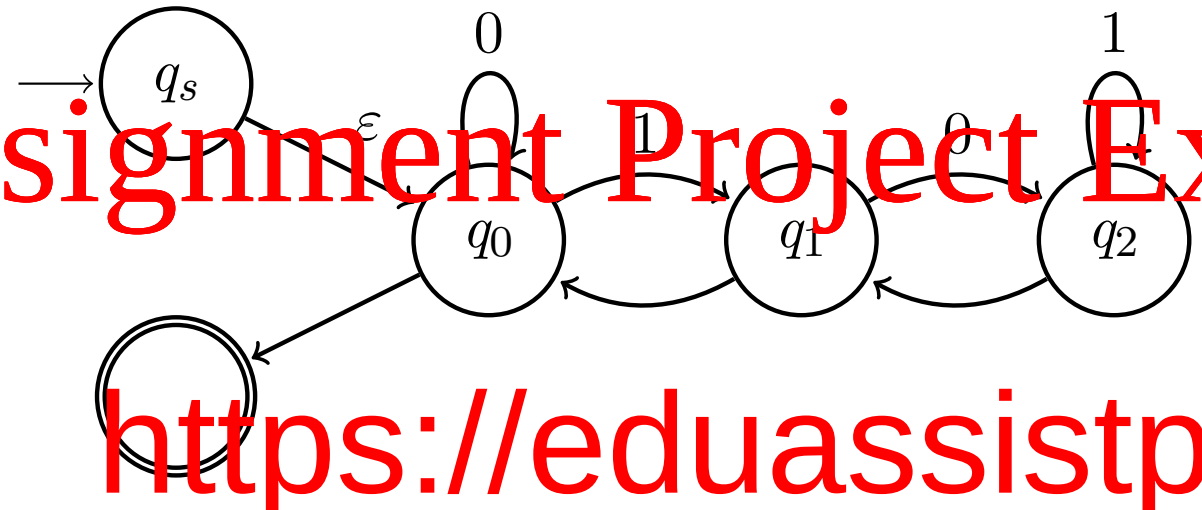


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) : R_1 = \varepsilon, R_2 = 0, R_3 = \varepsilon, R_4 = \emptyset \Rightarrow \varepsilon 0^\star \varepsilon \mid \emptyset$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

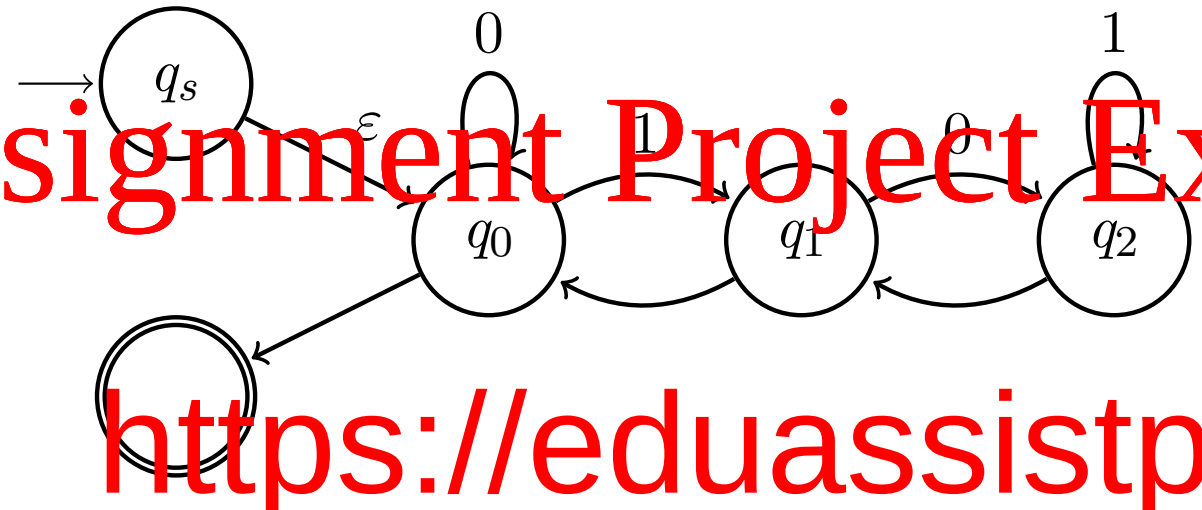


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) : R_1 = \varepsilon, R_2 = 0, R_3 = \varepsilon, R_4 = \emptyset \Rightarrow \varepsilon 0^\star \varepsilon \mid \emptyset = 0^\star$
- $(q_1, q_1) :$
- $(q_1, q_a) :$

Example 2: divisible by 3

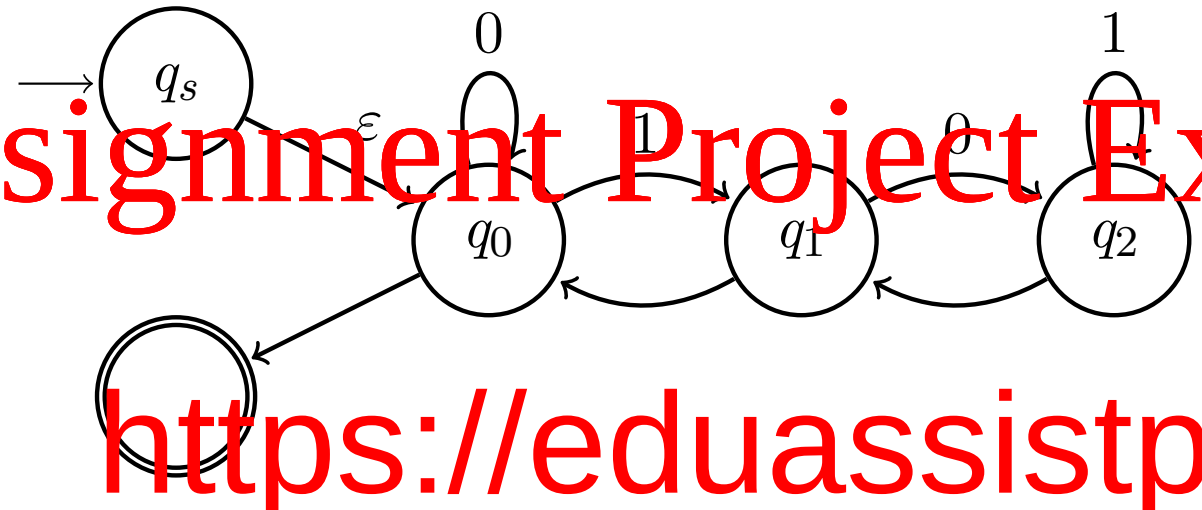


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) : R_1 = \varepsilon, R_2 = 0, R_3 = \varepsilon, R_4 = \emptyset \Rightarrow \varepsilon 0^\star \varepsilon \mid \emptyset = 0^\star$
- $(q_1, q_1) : R_1 = 1, R_2 = 0, R_3 = 1, R_4 = \emptyset \Rightarrow 10^\star 1 \mid \emptyset = 10^\star 1$
- $(q_1, q_a) :$

Example 2: divisible by 3

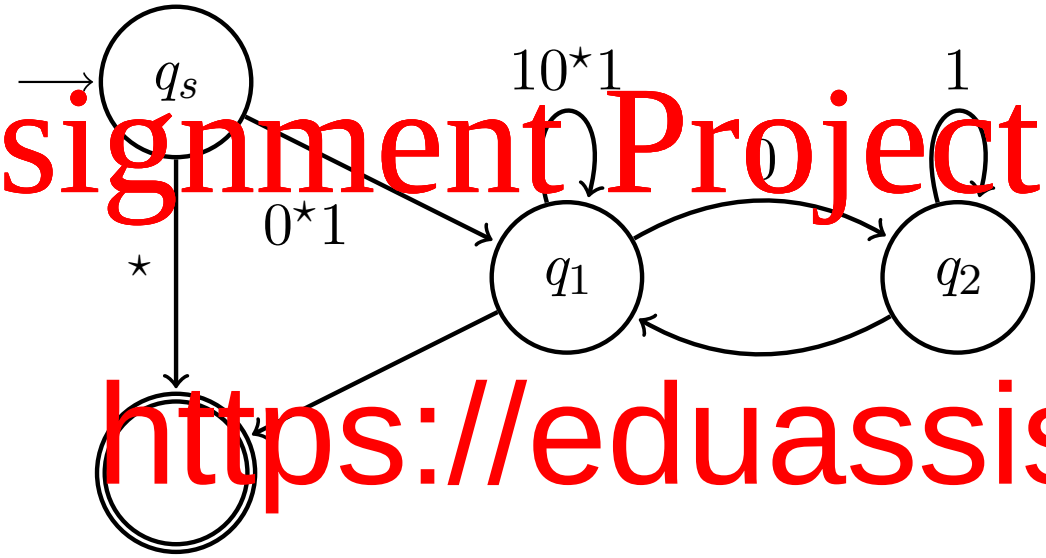


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not

- $(q_s, q_1) : R_1 = \varepsilon, R_2 = 0, R_3 = 1,$
- $(q_s, q_a) : R_1 = \varepsilon, R_2 = 0, R_3 = \varepsilon, R_4 = \emptyset \Rightarrow \varepsilon 0^\star \varepsilon \mid \emptyset = 0^\star$
- $(q_1, q_1) : R_1 = 1, R_2 = 0, R_3 = 1, R_4 = \emptyset \Rightarrow 10^\star 1 \mid \emptyset = 10^\star 1$
- $(q_1, q_a) : R_1 = 1, R_2 = 0, R_3 = \varepsilon, R_4 = \emptyset \Rightarrow 10^\star \varepsilon \mid \emptyset = 10^\star$

Example 2: divisible by 3

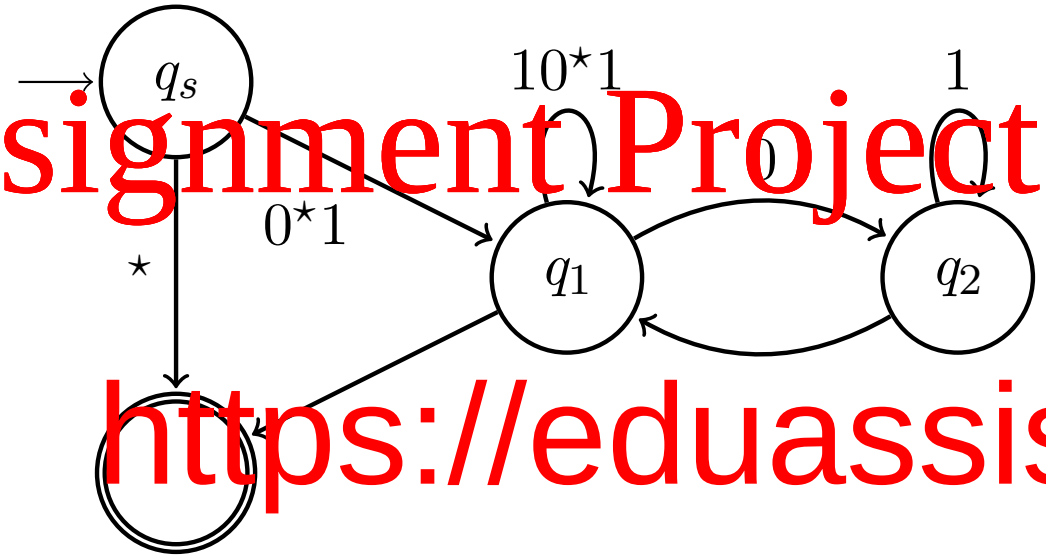


We can eliminate the states in any order. Eliminate

All pairs of states with transitions which are not 2

- ▶

Example 2: divisible by 3

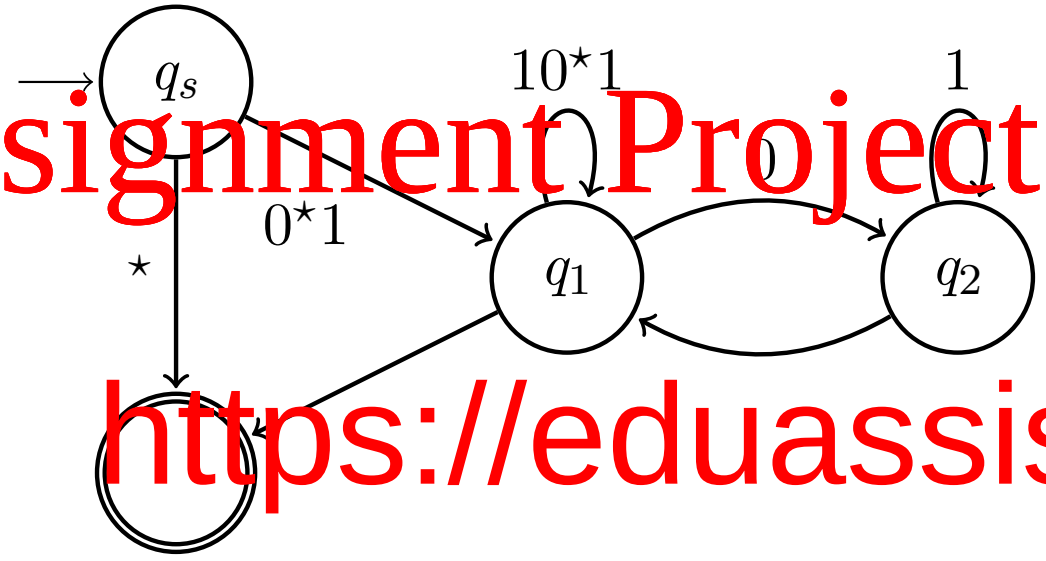


We can eliminate the states in any order. Eliminate q_2.

All pairs of states with transitions which are not q_2:

- (q_1, q_1) :

Example 2: divisible by 3

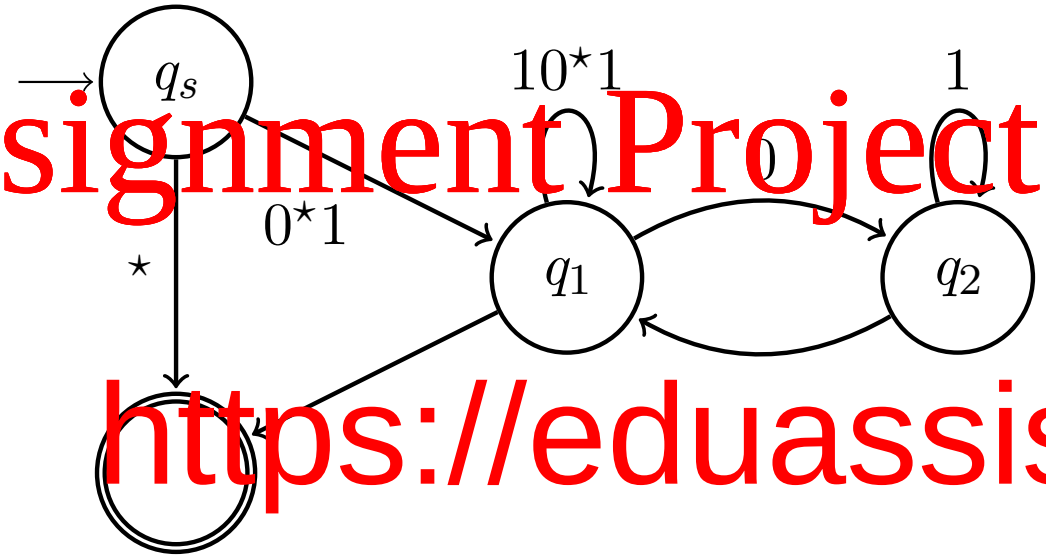


We can eliminate the states in any order. Eliminate q_1 first.

All pairs of states with transitions which are not q_2

- $(q_1, q_1) : R_1 = 0,$

Example 2: divisible by 3

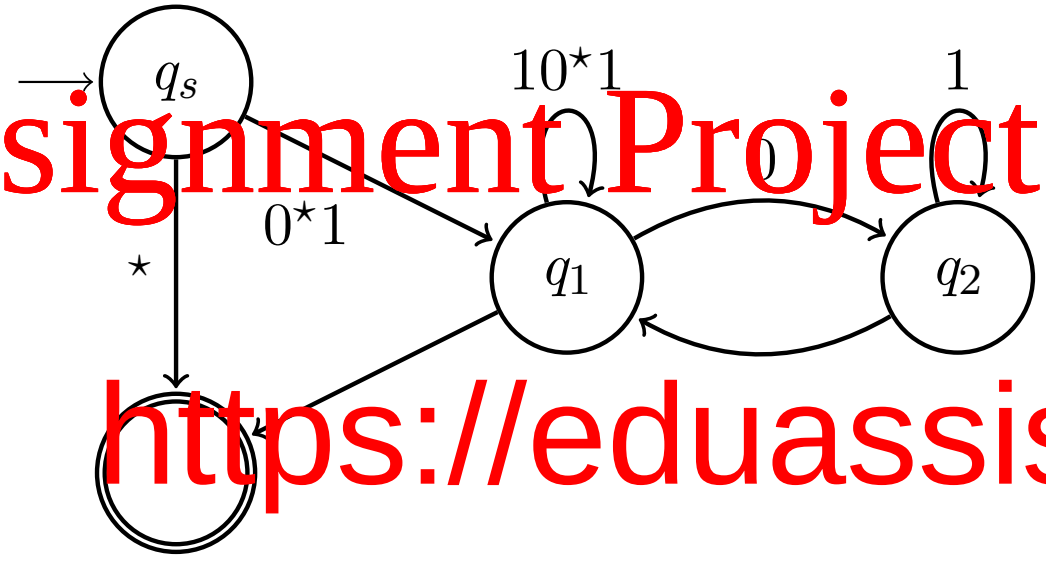


We can eliminate the states in any order. Eliminate q_2

All pairs of states with transitions which are not

- $(q_1, q_1) : R_1 = 0, R_2 = 1,$

Example 2: divisible by 3

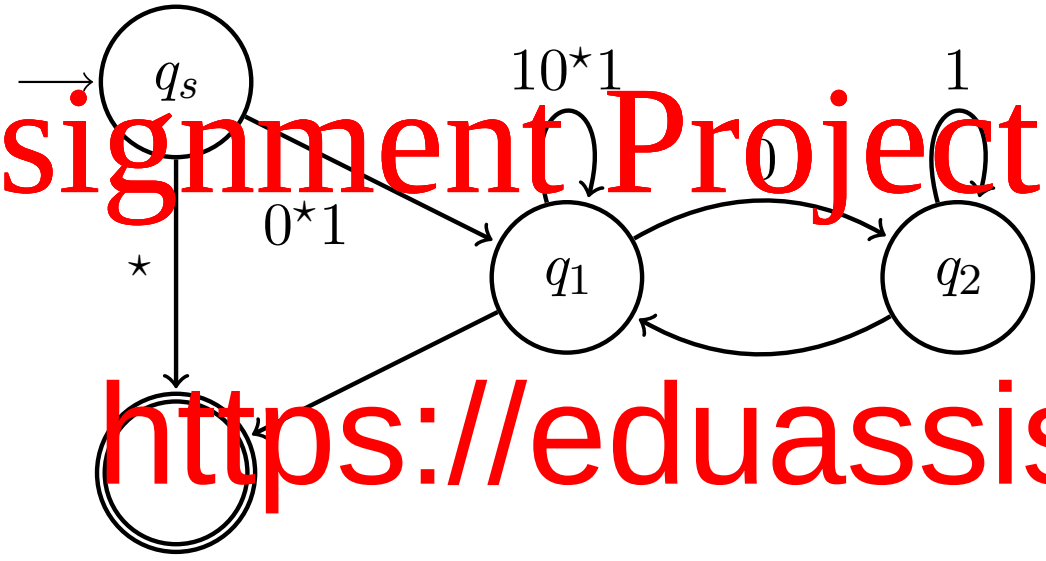


We can eliminate the states in any order. Eliminate q_2 first.

All pairs of states with transitions which are not

- $(q_1, q_1) : R_1 = 0, R_2 = 1, R_3 = 0,$

Example 2: divisible by 3

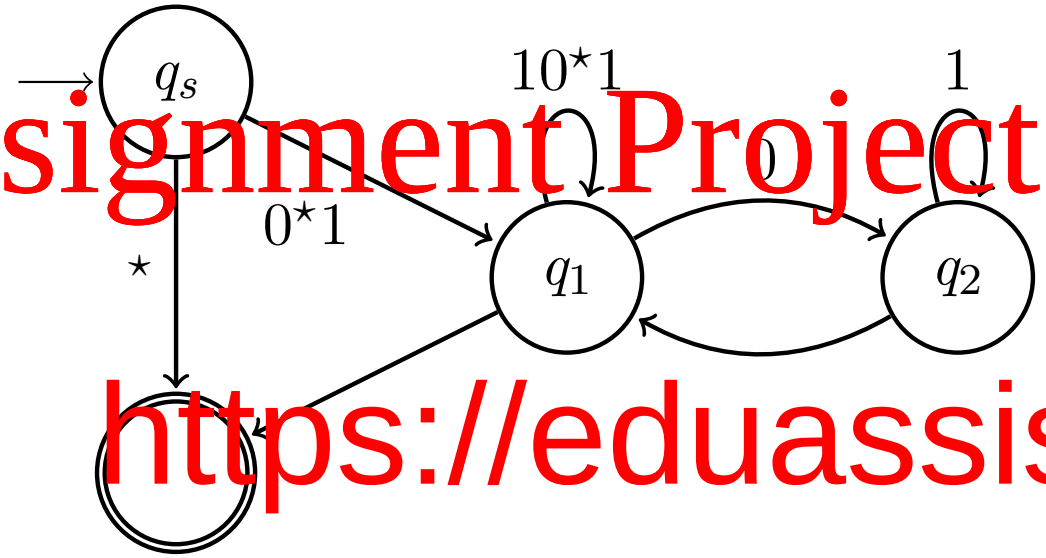


We can eliminate the states in any order. Eliminate q_2

All pairs of states with transitions which are not

- $(q_1, q_1) : R_1 = 0, R_2 = 1, R_3 = 0, R_4 = 10^\star1$

Example 2: divisible by 3



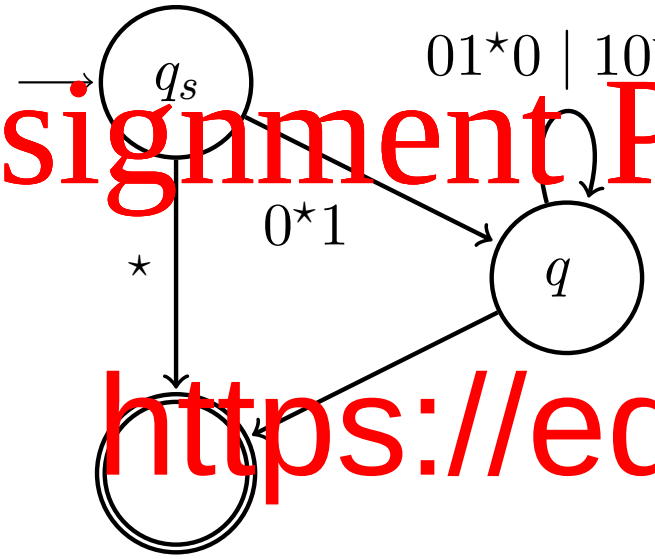
We can eliminate the states in any order. Eliminate 2

All pairs of states with transitions which are not

- $(q_1, q_1) : R_1 = 0, R_2 = 1, R_3 = 0, R_4 = 10^\star1 \Rightarrow 01^\star0 \mid 10^\star1$

Much easier!

Example 2: divisible by 3

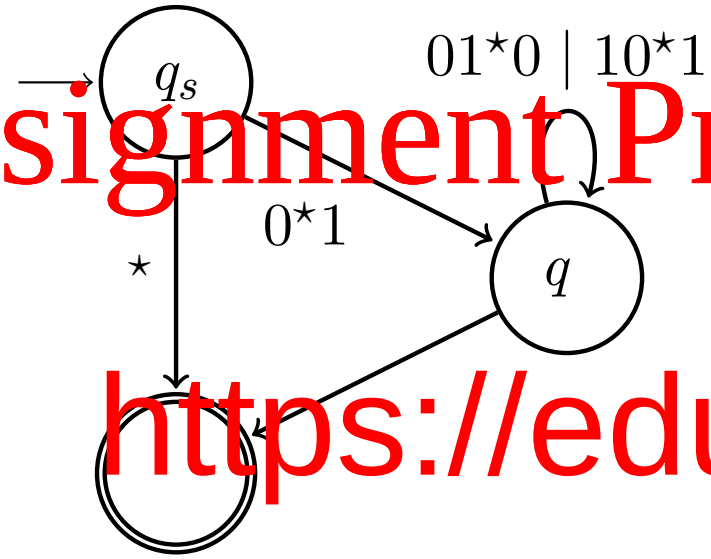


There is only q_1 left to remove.

All pairs of states with transitions which are not $\emptyset$ q_1

- ▶

Example 2: divisible by 3

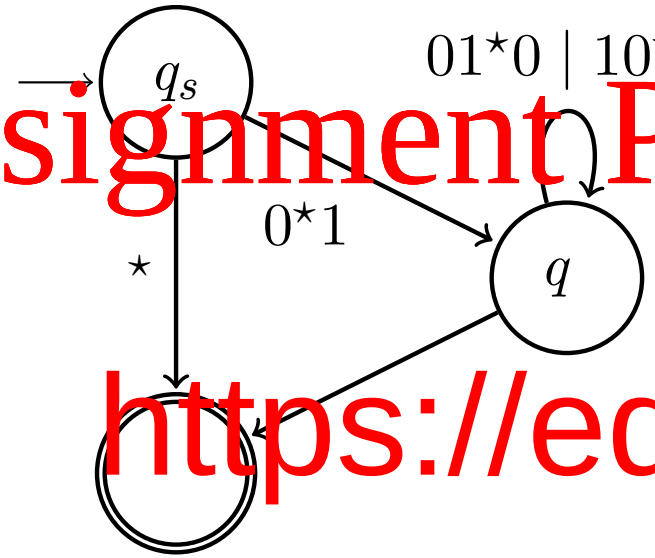


There is only q_1 left to remove.

All pairs of states with transitions which are not $\emptyset$ q_1

- (q_s, q_a) :

Example 2: divisible by 3

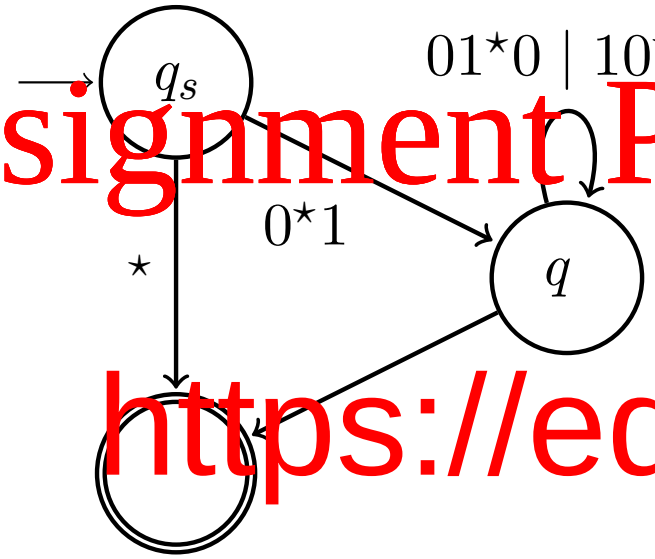


There is only q_1 left to remove.

All pairs of states with transitions which are not $\emptyset$ q_1

- $(q_s, q_a) : R_1 = 0^\star1,$

Example 2: divisible by 3

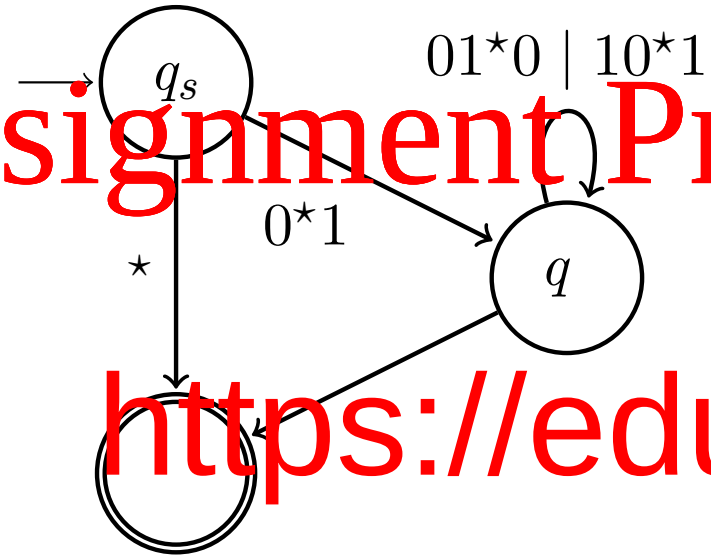


There is only q_1 left to remove.

All pairs of states with transitions which are not $\emptyset$ q_1

- $(q_s, q_a) : R_1 = 0^\star1, R_2 = 01^\star0 \mid 10^\star1,$

Example 2: divisible by 3

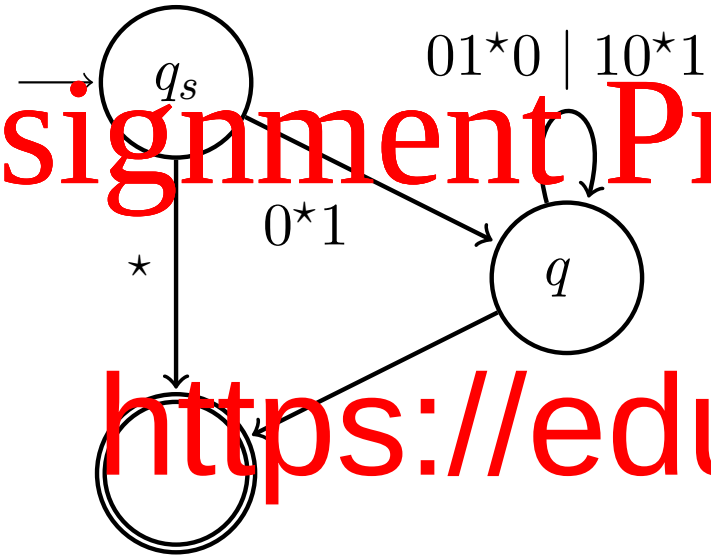


There is only q_1 left to remove.

All pairs of states with transitions which are not $\emptyset$ q_1

- $(q_s, q_a) : R_1 = 0^\star1, R_2 = 01^\star0 \mid 10^\star1, R_3 = 10^\star,$

Example 2: divisible by 3

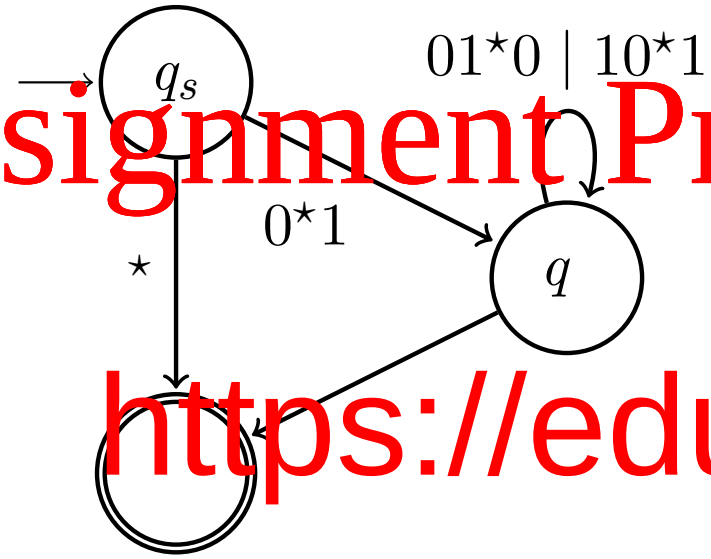


There is only q_1 left to remove.

All pairs of states with transitions which are not $\emptyset$ q_1

- $(q_s, q_a) : R_1 = 0^\star1, R_2 = 01^\star0 \mid 10^\star1, R_3 = 10^\star, R_4 = 0^\star$

Example 2: divisible by 3

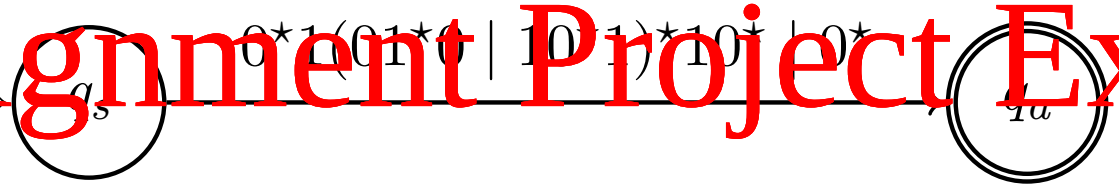


There is only q_1 left to remove.

All pairs of states with transitions which are not $\emptyset$ q_1

- $(q_s, q_a) : R_1 = 0^\star1, R_2 = 01^\star0 \mid 10^\star1, R_3 = 10^\star, R_4 = 0^\star$
  so $(R_1)(R_2)^\star(R_3) \mid (R_4) = 0^\star1(01^\star0 \mid 10^\star1)^\star10^\star \mid 0^\star$

Example 2: divisible by 3



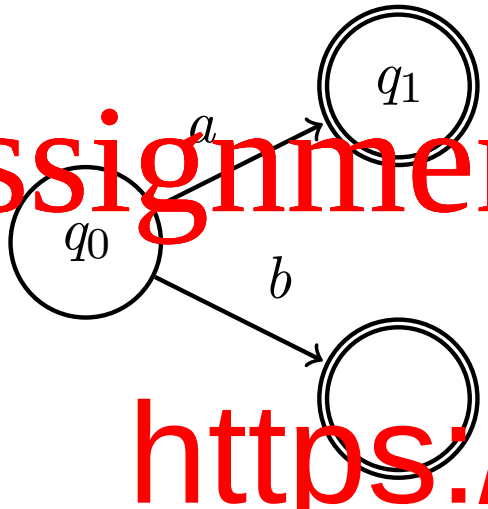
So,
strin

Example 2: divisible by 3

So,
strin

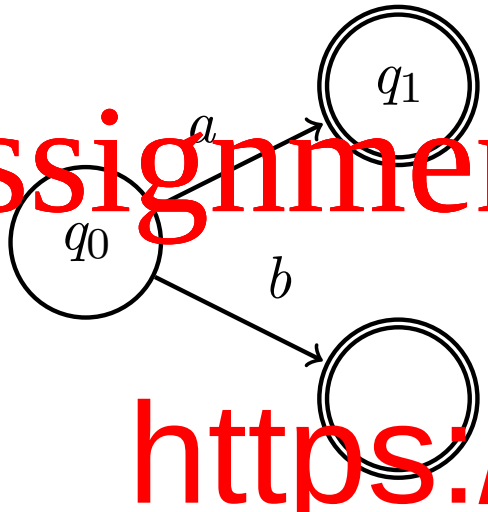
Eliminating states in a different order can result in a d
equivalent RegEx.
e.g. eliminating q_2, q_1, then q_0 yields: $(1(01^\star0)^\star1|0)^\star$

Example 3: Multiple accept states

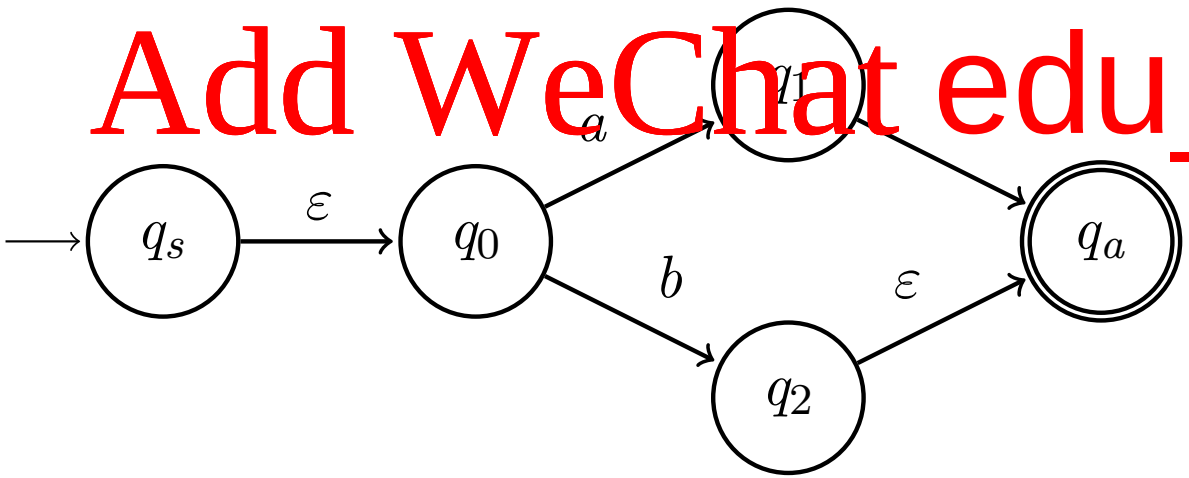


As GNFA:

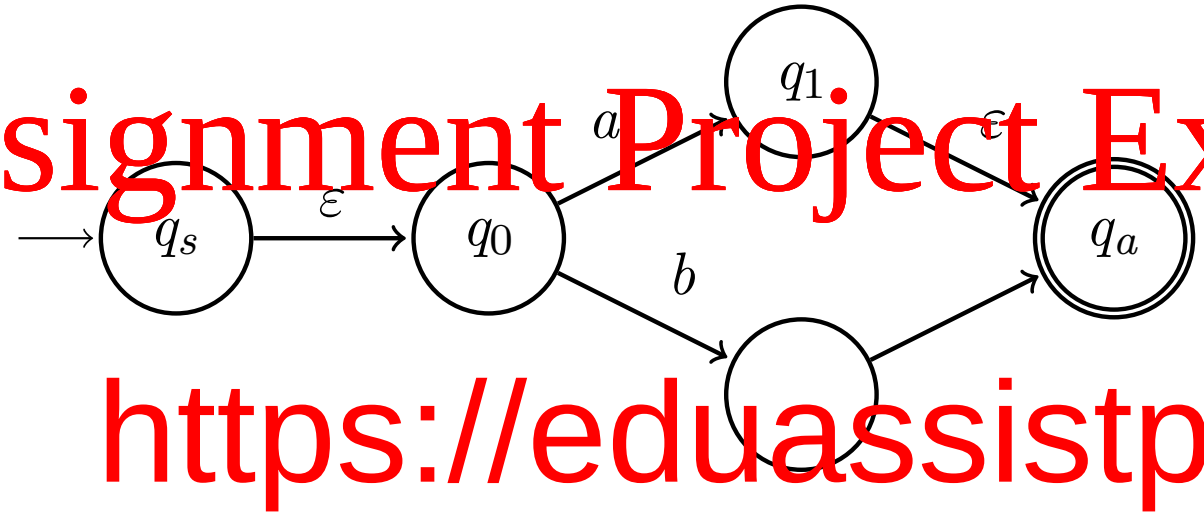
Example 3: Multiple accept states



As GNFA:

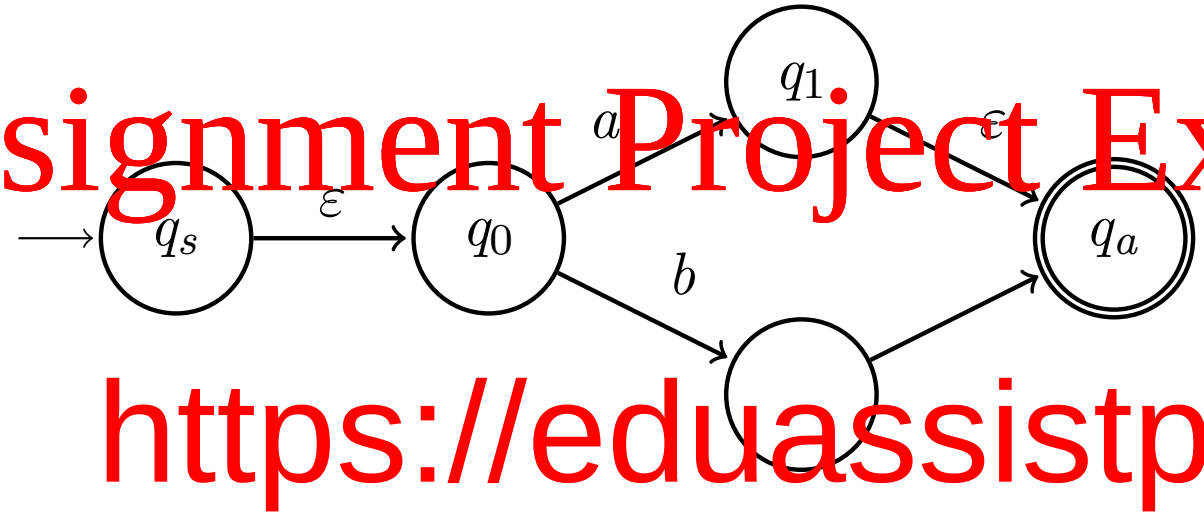


Example 3: Multiple accept states

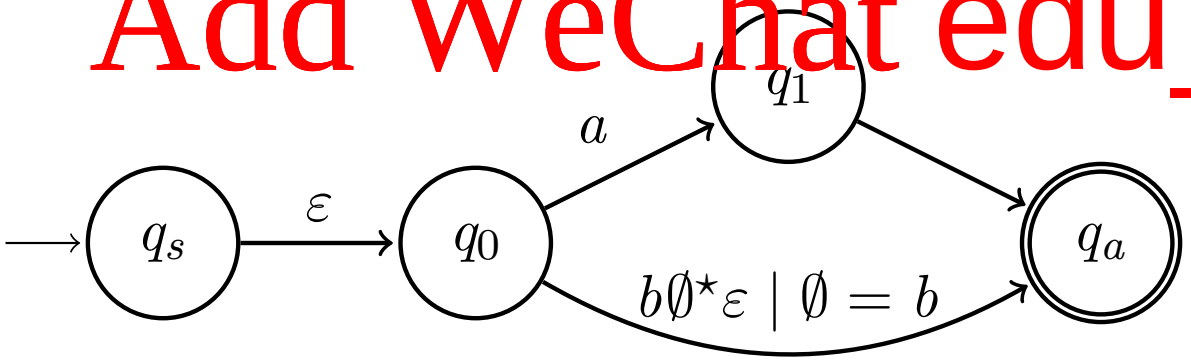


Eliminate q_2:

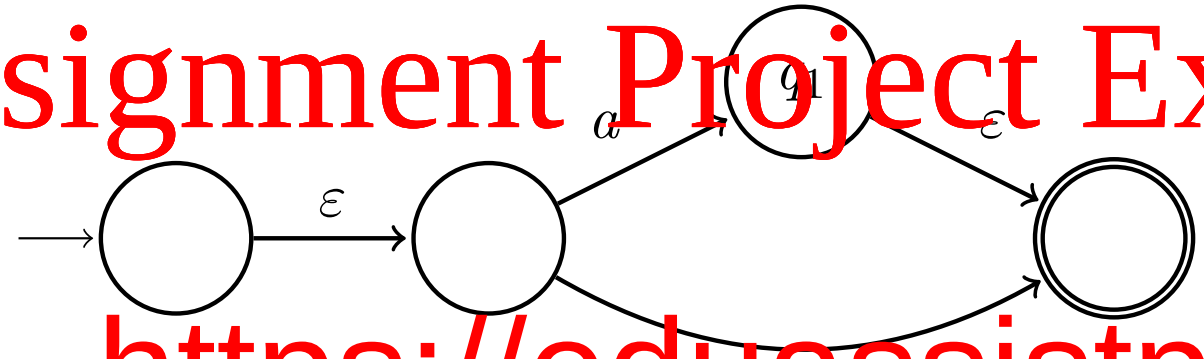
Example 3: Multiple accept states



Eliminate q_2:

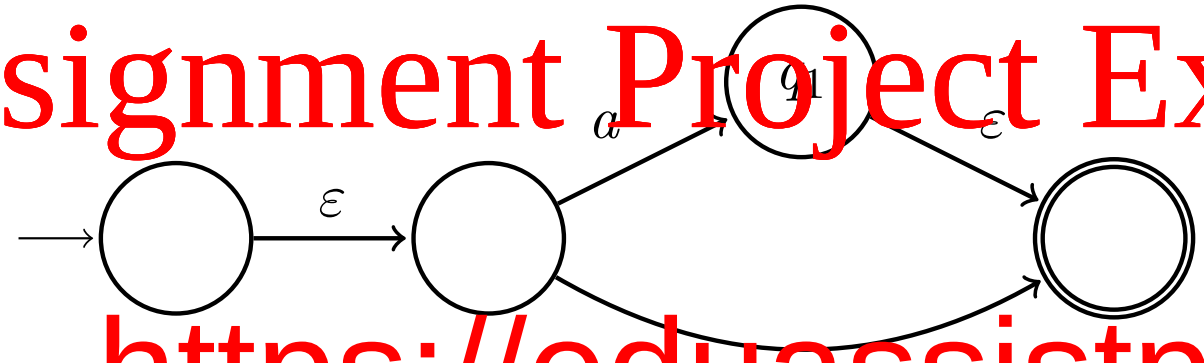


Example 3: Multiple accept states

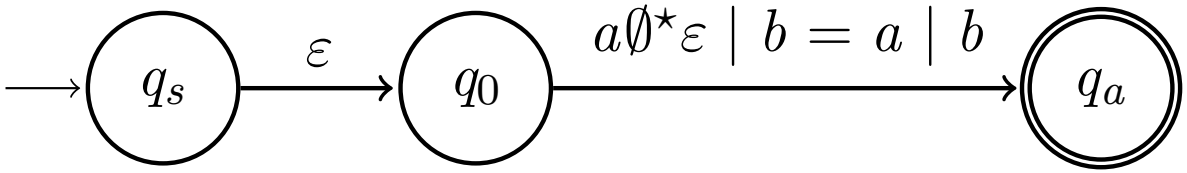


Eliminate q_1:

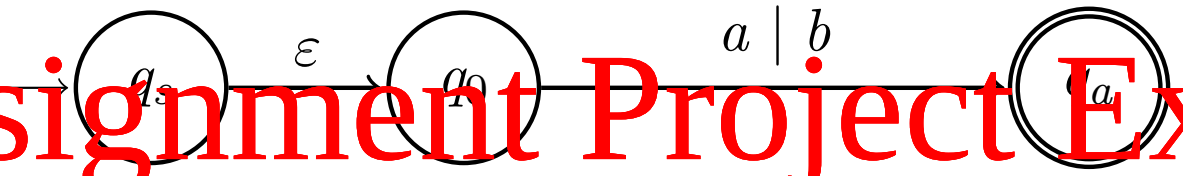
Example 3: Multiple accept states



Eliminate q_1:



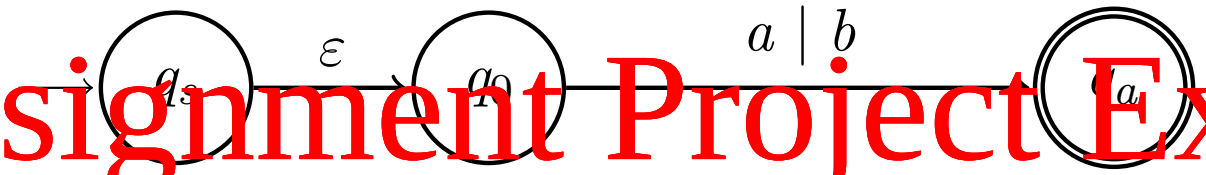
Example 3: Multiple accept states



Elim

Simplify, to get the expected.

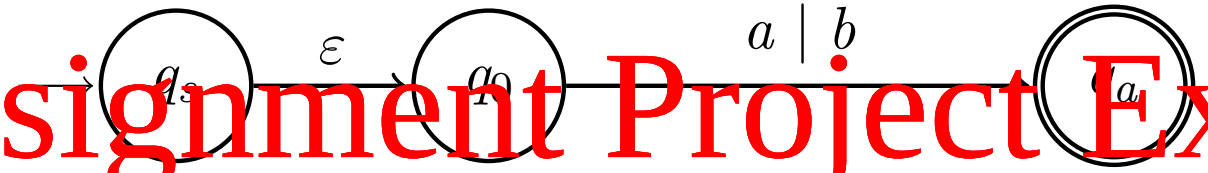
Example 3: Multiple accept states



Elim

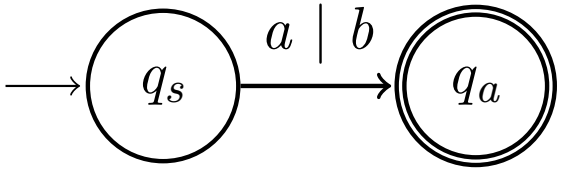
Simplify, to get the expected.

Example 3: Multiple accept states



Elim

Simplify, to get the expected:



Summary

Algorithm to convert an NFA to a RegEx:

1. Convert the NFA to a Generalised NFA (GNFA)
   - Only one start state, no incoming transitions

Summary

Algorithm to convert an NFA to a RegEx:

1. Convert the NFA to a Generalised NFA (GNFA)
   - Only one start state, no incoming transitions
2. (q_i, q_j) q_i, q_j Q q_{elim}
   - Replace arrow $q_i \to q_j$ with $((R$
     - R_1 is the RegEx on transition
     - R_2 is the RegEx on transition
     - R_3 is the RegEx on transition $_{elim}$ $_{j}$
     - R_4 is the RegEx on transition $q_i \to q_j$

Summary

Algorithm to convert an NFA to a RegEx:

1. Convert the NFA to a Generalised NFA (GNFA)
   - Only one start state, no incoming transitions
2. For each pair of states (q_i, q_j) other than the accept and start state q_i, q_j from Q other than q_{elim}
   - Replace arrow $q_i \to q_j$ with $((R_1 R_2^* R_3) \cup R_4)$
     - R_1 is the RegEx on transition $q_i \to q_{elim}$
     - R_2 is the RegEx on transition $q_{elim} \to q_{elim}$
     - R_3 is the RegEx on transition $q_{elim} \to q_j$
     - R_4 is the RegEx on transition $q_i \to q_j$
3. The transition between the start and the accept state is now a regular expression describing L

Outline

- Regular Expressions
-
-
- **Proving if a language is, *or is not***

Proving if a language is regular *or not*

Recall the definition:

A language is *regular* if and only if there exists a finite automaton which recognises it.

Supp

Proving if a language is regular *or not*

Recall the definition:

A language is *regular* if and only if there exists a finite automaton which recognises it.

Suppose we want to prove that a language is regular. We can:

- Devise a DFA which recognises it, or
- Devise a RegEx which describes it

Proving if a language is regular *or not*

Recall the definition:

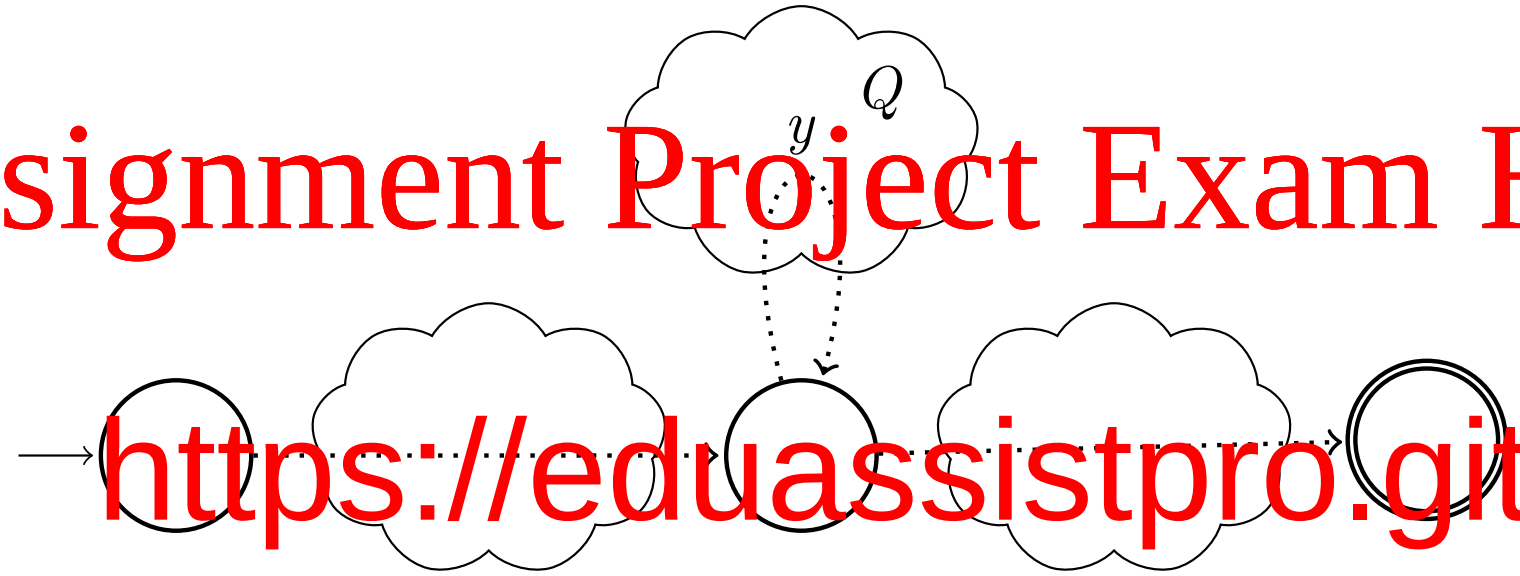
A language is *regular* if and only if there exists a finite automaton which recognises it.

Suppose we want to prove a language is regular. We can:

- Devise a DFA which recognises it, or
- Devise a RegEx which describes it

What if the language was not regular? How can we *prove* that no suitable automata exists? We can use the *pumping lemma for regular langauges*

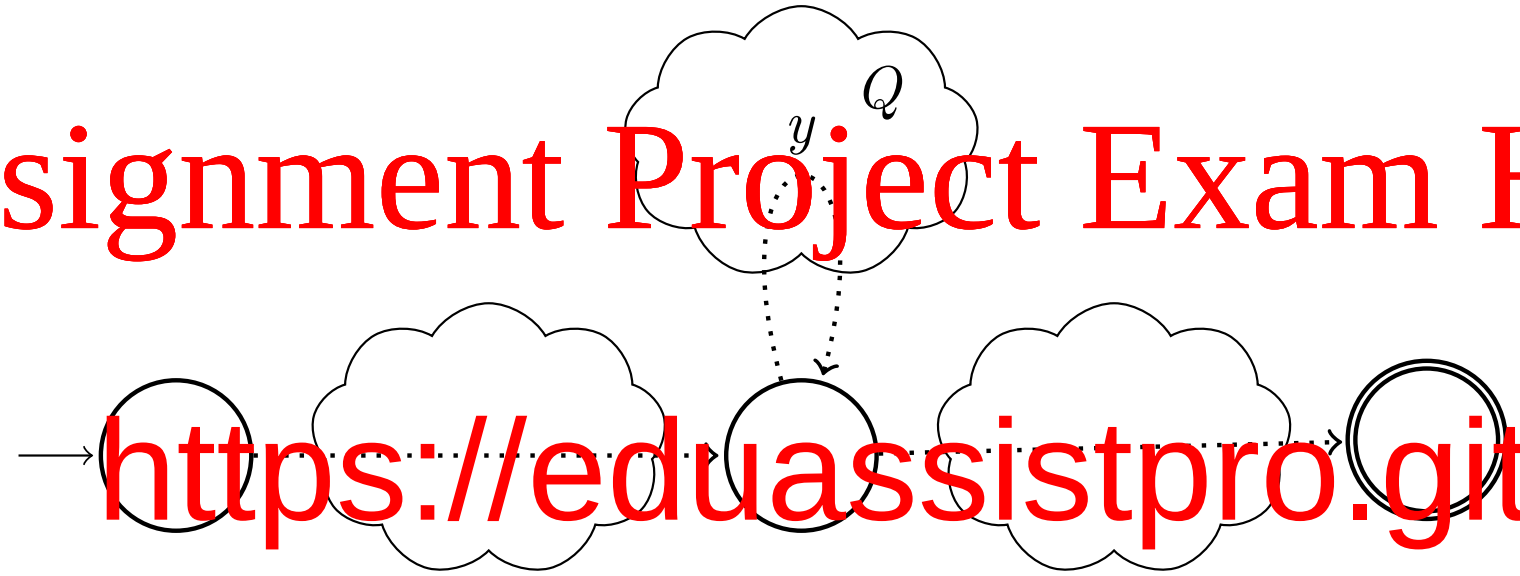
Pumping s using y



Suppose some string s exists, passing thr

- From q_0 to q_i using a string x, the
- from q_i back to q_i, using a string /
- from q_i to some accept state $f \in F$, using a string z

Pumping s using y



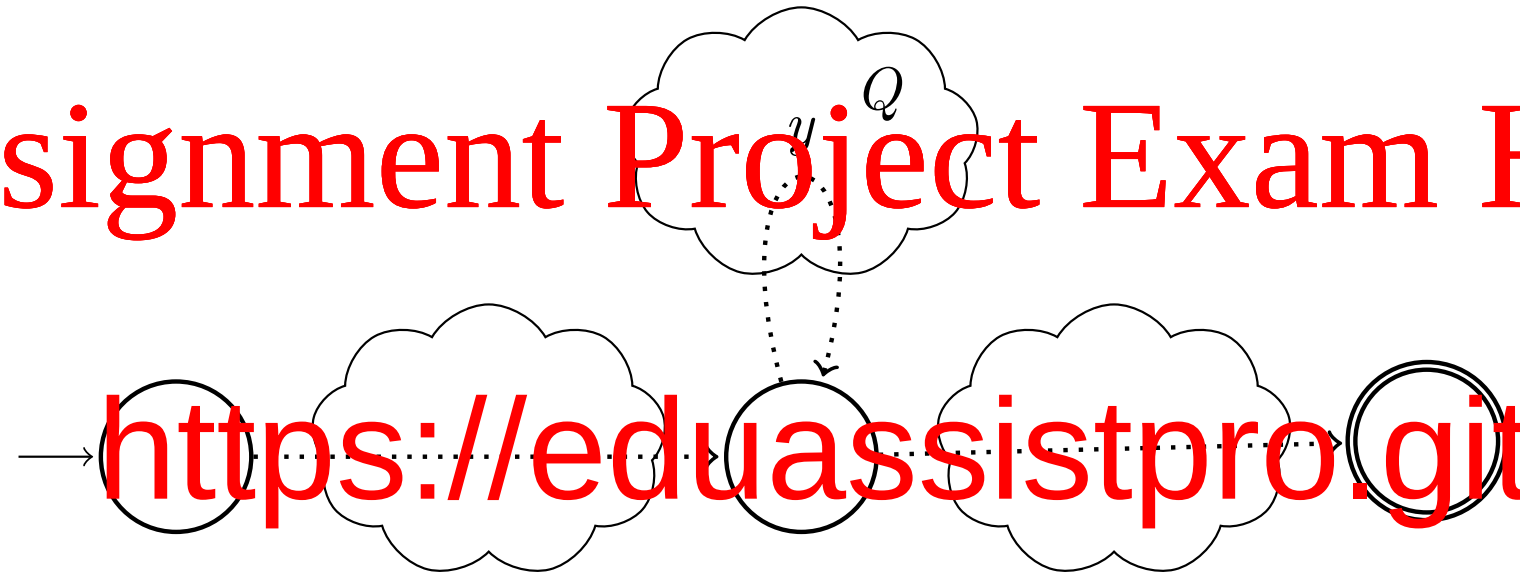
Suppose some string s exists, passing thr

- From q_0 to q_i, using a string x, the
- from q_i back to q_i, using a string /
- from q_i to some accept state $f \in F$, using a string z

Then $s = xyz \in L$, and $xy^k z \in L$ for all $k \geq 0$

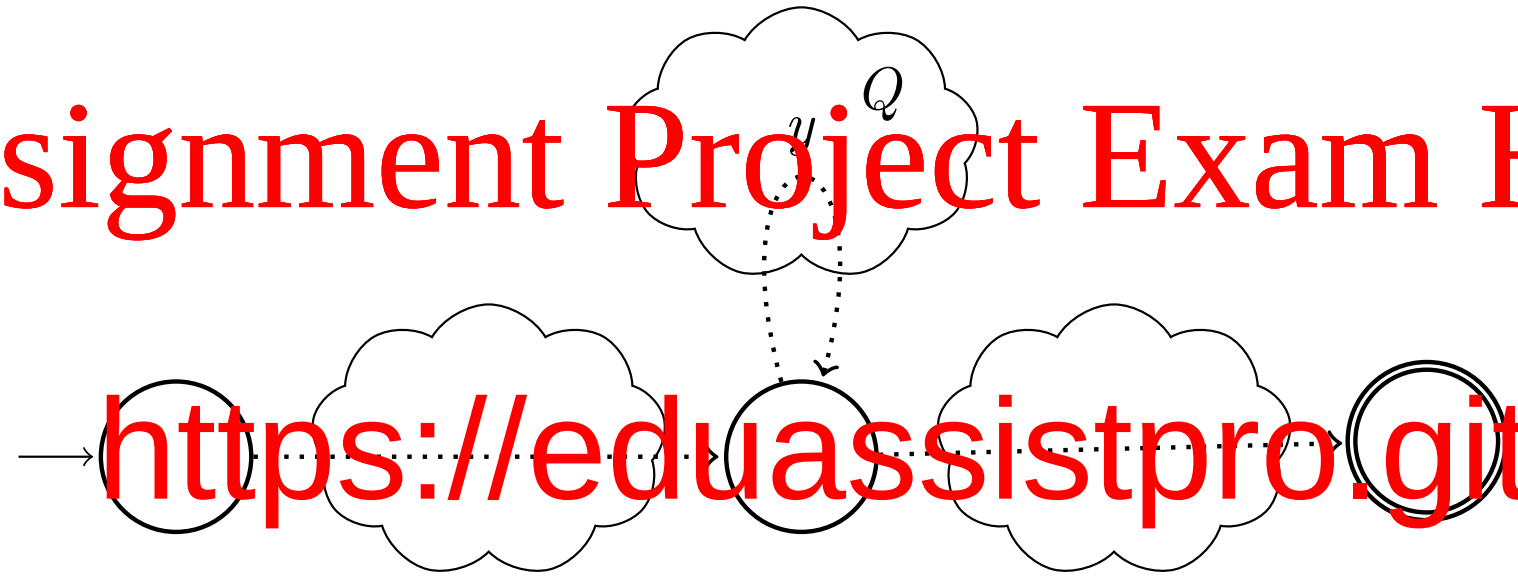
e.g. if $x = aa, y = b, z = c$, then $\{aac, aabc, aabbc, ...\} \subseteq L$

Long strings in finite automata



Suppose M is a DFA with n states, and $+1$

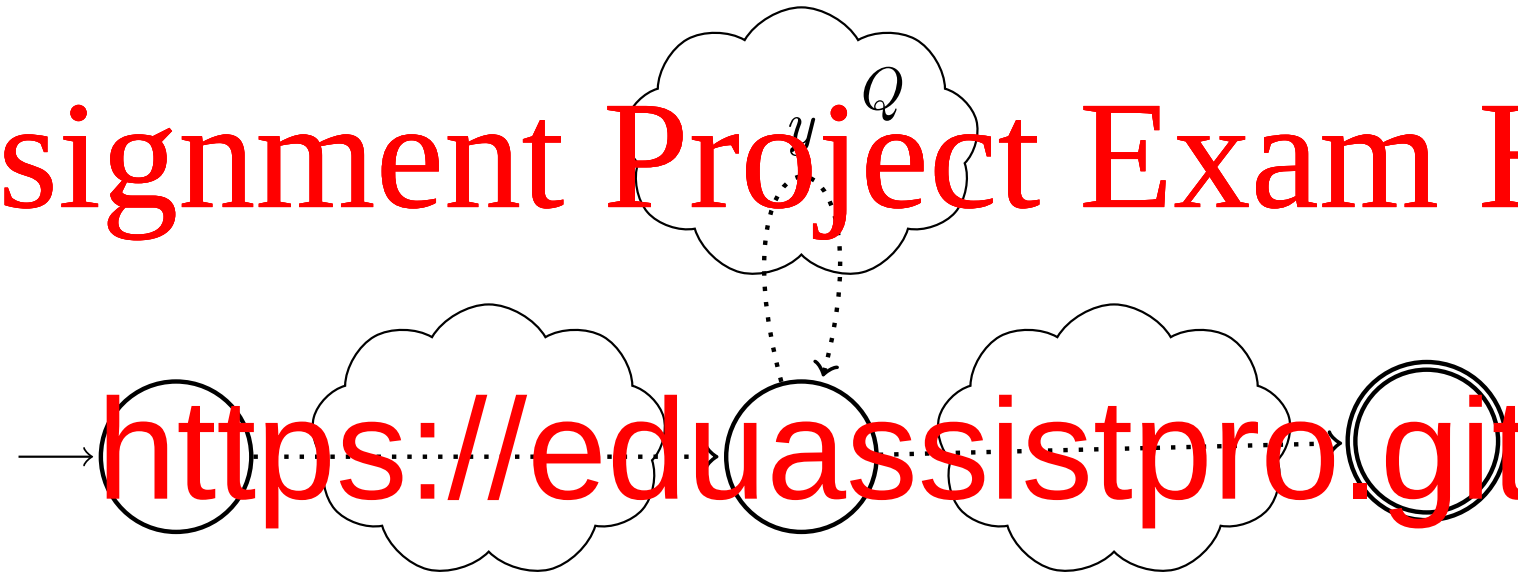
Long strings in finite automata



Suppose M is a DFA with n states, and ≥ 1

Then there exists at least one state which was visite
once (q_i), and a substring $y \neq \varepsilon$ corresponding to the path followed between the two of those visits.

Long strings in finite automata



Suppose M is a DFA with n states, and ≥ 1

Then there exists at least one state which was visite once (q_i), and a substring $y \neq \varepsilon$ corresponding to the path followed between the two of those visits.

Hence we can *pump* s to find other strings in the language

Finite automata, infinite languages

All finite languages are regular. (Why?)

Finite automata, infinite languages

All finite languages are regular. (Why?)

Supp

- L M
- L is an infinite set over a finite alphabet, so it mu arbitrarily long words.
- Therefore words exist which can be

Pumping lemma for regular languages

If L is a regular language, then there exists a number p (the *pumping length*) such that for any string $s \in L$ of length at least p, the

1.
2.
3. $|xy| \leq p$

If a language does not satisfy this lemma, then it cannot be regular

Using the pumping lemma

We can (only) use the pumping lemma to prove that a language is *not* regular

Proo

1.

2.
    - This is often the hardest part of the proof
    - We must find one which will allow us to:

3. Apply the lemma to find a contradiction
4. Thereby deduce that the assumption is false
    - i.e. The language cannot be regular

Example

Show that $L = \{a^n b^n | n \geq 0\}$ is not regular

Example

Show that $L = \{a^n b^n | n \geq 0\}$ is not regular

1. Assumption: L is regular.

Example

Show that $L = \{a^n b^n | n \geq 0\}$ is not regular

1. Assumption: L is regular.
2. Then there exists some p 0 satisfying the pumping lemma.

Example

Show that $L = \{a^n b^n | n \geq 0\}$ is not regular

1. Assumption: L is regular.
2. Then there exists some p 0 satisfying the pumping lemma.
3.

Example

Show that $L = \{a^n b^n | n > 0\}$ is not regular

1. Assumption: L is regular.
2. Then there exists some p 0 satisfying the pumping lemma.
3.
4.
   - 4.2 $|y| > 0$
   - 4.3 $|xy| \leq p$

Example

Show that $L = \{a^n b^n | n > 0\}$ is not regular

1. Assumption: L is regular.
2. Then there exists some p $\quad 0$ satisfying the pumping lemma.
3.
4.
   - 4.2 $|y| > 0$
   - 4.3 $|xy| \leq p$
5. Therefore $y = a^i$ for some $0 < i$ s)

Example

Show that $L = \{a^n b^n | n > 0\}$ is not regular

1. Assumption: L is regular.
2. Then there exists some p 0 satisfying the pumping lemma.
3.
4.
   - 4.2 $|y| > 0$
   - 4.3 $|xy| \leq p$
5. Therefore $y = a^i$ for some $0 <$ i s)
6. Let $k = 2$. Then $xy^k z$ has more a b xy $z \notin L$

Example

Show that $L = \{a^n b^n | n > 0\}$ is not regular

1. Assumption: L is regular.
2. Then there exists some p 0 satisfying the pumping lemma.
3.
4.
   - 4.2 $|y| > 0$
   - 4.3 $|xy| \leq p$
5. Therefore $y = a^i$ for some $0 <$ i s)
6. Let $k = 2$. Then $xy^k z$ has more a b xy $z \notin L$
7. Lines 6 and 4.1 form a contradiction, therefore the assumption is false (i.e. L is *not* regular.)

Introduction

So far we have seen two different, but equivalent, methods of describing languages: finite automata and regular expressions, which describe *regular languages*

We ha
$\{0^n 1$

Today we will introduce *context-free gr*
describe the next category of languages, the *es*

Later, will see grammars called *regular grammars*, which describe exactly *regular languages*

Grammars

Grammars are another way to describe a language

A *gr* a
lang

The language generated is the set of all strings whic
derived from the grammar

Introductory example (G_1)

$S \to 01$

Base case: $01 \in L$

L

G_1 g
kno

dy

How does it derive 000111?

Introductory example (G_1)

$S \to 01$

Base case: $01 \in L$

L

G_1 g
kno

dy

How does it derive 000111?

S

Introductory example (G_1)

$S \to 01$

Base case: $01 \in L$

L

G_1 g

kno

dy

How does it derive 000111?

$S \Rightarrow 0S1$

usi

Introductory example (G_1)

$S \to 01$

Base case: $01 \in L$

L

G_1 g
kno

dy

How does it derive 000111?

$S \Rightarrow 0S1$

usi

$\Rightarrow 00S11$

using rule $S \to 0S1$

Introductory example (G_1)

$S \to 01$

Base case: $01 \in L$

L

G_1 g

kno

dy

How does it derive 000111?

$S \Rightarrow 0S1$ usi

$\Rightarrow 00S11$ using rule $S \to 0S1$

$\Rightarrow 000111$ using rule $S \to 01$

Introductory example (G_2)

$$S \to NounPhrase\ VerbPhrase$$

$$Verb \to \text{likes} \mid \text{see}$$

What language does G_2 generate?

Introductory example (G_2)

$$S \to NounPhrase\ VerbPhrase$$

$$Verb \to \text{likes} \mid \text{see}$$

What language does G_2 generate?

{ the girl likes the girl, the girl likes the ball,
the girl sees the girl, the girl sees the ball,
the ball likes the girl, the ball likes the ball,
the ball sees the girl, the ball sees the ball }

Definitions

Terminals

- The finite set of symbols which make up strings of the language

Non-

- A finite set of symbols used to generate the stri
- They never appear in the language.

Start symbol

- The variable used to start every derivation

Definitions

Production rules

- Sometimes called substitution or derivation rules
- Define strings of *variables* and *terminals* which can be

Definitions

Production rules

- Sometimes called substitution or derivation rules
- Define strings of *variables* and *terminals* which can be

A variable can have many rules:

$Noun \rightarrow$ girl

$Noun \rightarrow$ ball

$Noun \rightarrow$ quokka

Definitions

Production rules

- Sometimes called substitution or derivation rules
- Define strings of *variables* and *terminals* which can be

A variable can have many rules: Th

$Noun \rightarrow$ girl ka

$Noun \rightarrow$ ball

$Noun \rightarrow$ quokka

Another example

$$S \to T \mid (S \cdot S) \mid (\lambda T.S)$$

This is a grammar for lambda calculus expression

- The variables are S, T
- S is the start variable
- The terminals are $a, b, ..., (,), \lambda, ., \cdot$ (i.e. atoms and operators)

Some common notational conventions

If not stated otherwise:

-
-
-
-
- ..., X, Y, Z are either terminals or varia
- ..., w, x, y, z are strings of terminals
- $\alpha, \beta, \gamma, \ldots$ are strings of terminals and/o

NIL

In your tree encoding, NIL should represent the empty tree (a tree with no nodes)

This is

If you need to use both and they are different, then you c rename them. e.g.

- $NIL = PAIR\ TRUE\ TRUE$, for Ch
- $NILTREE$, for your tree encoding

Example answer

$(HEAD\ z)$ should be the head of list z

$HEAD = \lambda z.FIRST\ (SECOND\ z)$

The second pair in a Church list contains the data $[head, tail]$, so we just need to get the first expression from the second pair.

Exa

$=$ tation)

$= HEAD\ (PAIR\ TRUE\ (PAIR\ (1$

$= (\lambda z.FIRST\ (SECOND\ z))\ (PAI$ $))$

$= FIRST\ (SECOND\ (PAIR\ TRUE\ (PAIR\ 1\ \{2,3\}))))$

$= FIRST\ (PAIR\ 1\ \{2,3\}))$ (reduced SECOND)

$= 1$ (reduced FIRST)

Immutable data structures

Example: Delete the second element of a list

- CONS to make a new list using:
- the head of the old list
- the tail of the tail of the old list
  - i.e. we skipped the second element

Example: Insert e to the second position of a list

$$INS\ (CONS\ x\ (TAIL\ z))$$

- CONS a new list using:
- the head of the old list
  - e
  - the tail of the old list

Recursive data structures

Linked Lists

- Each position has a reference (link) to the next one

Chur

they link to the list which *starts with* t

- CONS head tail
  - head is the element stored at the start of the li
  - tail is the sublist containing all the remaining elements

Recursive data structures

You can implement a tree data structure in a very similar way.

Exce
and a n

In your assignment, *MAKETREE* nee
which will take a number to store, and two trees (whi
left and right children).

Java example

```
public class Tree {
    public final int element;
    public final Tree left;
    p

    p

        this.left = left;
        this.right = right;
    }
}
```


MAKETREE in the assignment is like writing: `new Tree(e,a,b);`

Recursive functions

Simple recursion, condition decides if you're at the base case

$$H = \lambda f x y z.(\text{condition})(\text{base})(\text{recursive})$$

$$F = (Y\ H)$$

You c

$$H \qquad\qquad)\ 3))$$

$$F = (Y\ H)$$

- If $x < 5$ and $y < 5$ then y (base ca
- If $x < 5$ and $y \geq 5$ then $f(1, x)$ (recursive case)
- If $x \geq 5$ and $x = y$ then $f(y, x)$ (recursive case)
- If $x \geq 5$ and $x \neq y$ then 3 (base case)

Helper functions

It's a good idea to define some extra expressions sometimes

-
-

Example: a MAX function which returns the large
would be helpful when calculating the height of a tre

LISP

I discourage you from trying to directly implement your lambda calculus expressions in LISP. It'll mostly work, but some things get messy (especially conditional logic) if you try to implement ever

I rec

- elements
- A Church list can be represented by a Lisp list

Then you can just use standard Lisp list operations to manipulate your data, instead of needing to implement all your lambda calculus expressions directly.

LISP

Don't forget to indent your code to make it easier to see what is being applied to what. Most errors people have shown me were just scope errors.

```
;; example of a
(def
  (if
    (...) ;; some condition
    (...) ;; true case
    (...) ;; false case
  )
)
```

LISP

```
;; example of a nested if statement
(defun foo (x y)
  (if
    (...)  ;; true 2
    (...)  ;; false 2
    )
   (...)   ;; false 1
  )
)
```

LISP

You don't need any notation we haven't provided in tutorials or lectu

Look back at the earlier tutorials and solutions for s examples.

Regular Expressions

- What they are and how to use them
- Regular operations

Equivalence of FA and Regular Expressions

-
-

Proving if a language is, *or is not*, regul

- Prove regularity by finding a DFA, NFA, or R
- Prove non-regularity by finding a contrad
  Pumping Lemma

Grammars (basic concepts)

Lambda calculus revision