COMP30026 Models of Computation

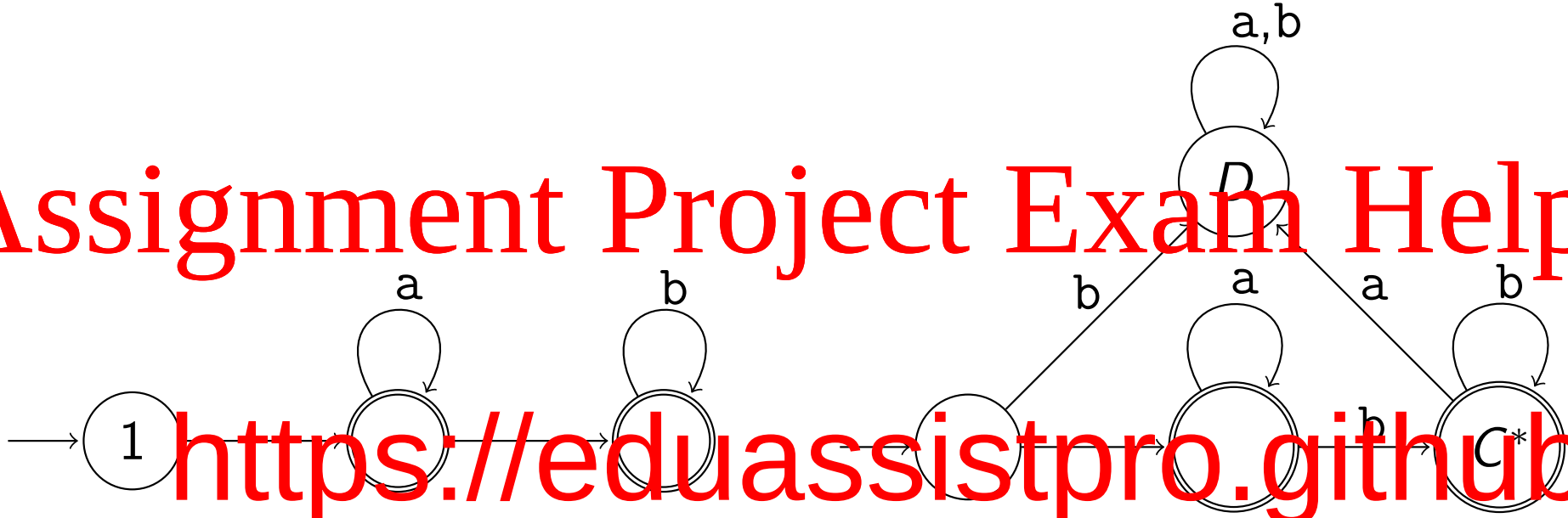
Regular Expressions and Non-Regular Languages

Lecture Week 8 Part

Semester 2, 2021

This Lecture is Being Recorded

Subset Construction Again...



Adding new state to DFA:

1. Step 1: Move on a symbol
2. Step 2: Build ϵ-closure

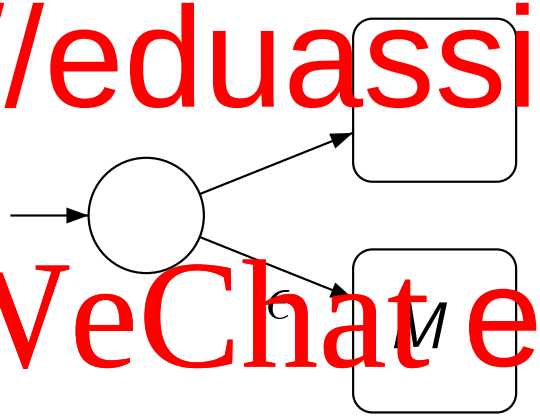
$C^*=\{3\}$	D	C^*
$D=\emptyset$	D	D

Closure Results for Regular Languages

Theorem: The class of regular languages is closed under union.

Proof: Let A and B be regular languages, with DFAs M_A and M_B as recognis

recognis

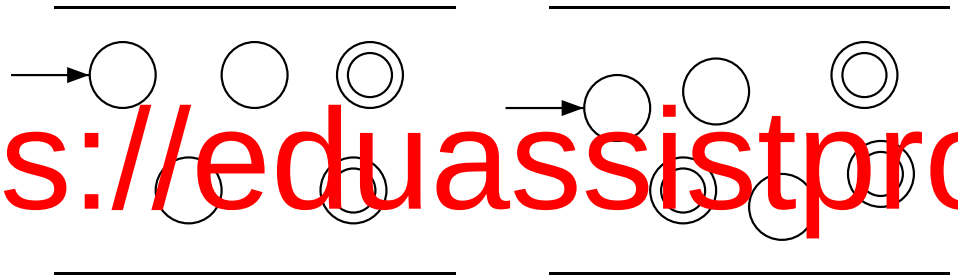


The ϵ-transitions go to the start states of M_A and M_B.

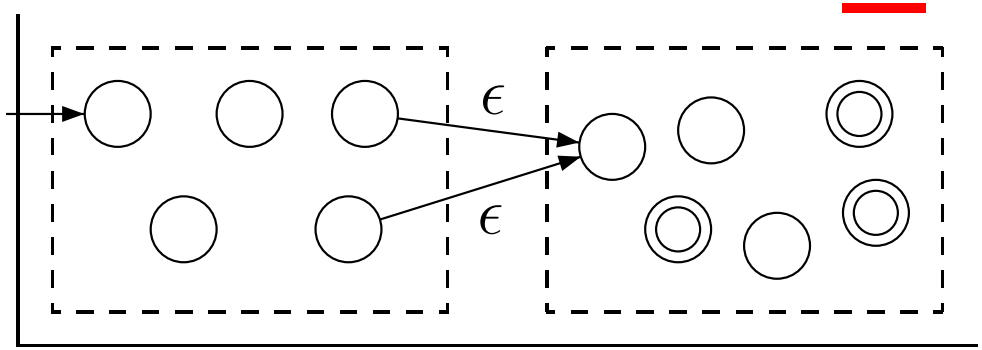
Closure Results for Regular Languages

Theorem: The class of regular languages is closed under $\circ$.

Proof: Let A and B be regular, with these DFAs as recognisers:



From these we can easily construct an NFA that reco $\circ B$:



That Last Construction, Formally

Let recognisers for A and B be these DFAs, respectively:

- $M_A = (Q, \Sigma, \delta, q_0, F)$
- M_B

A recogni $_0$), where

$$\delta'(q, v) = \begin{cases} \{\delta'(q, v)\} & \text{if} \\ \{\delta(q, v)\} & \text{if} \\ \{q_0'\} & \text{if } q \in F \text{ and } v = \epsilon \end{cases}$$

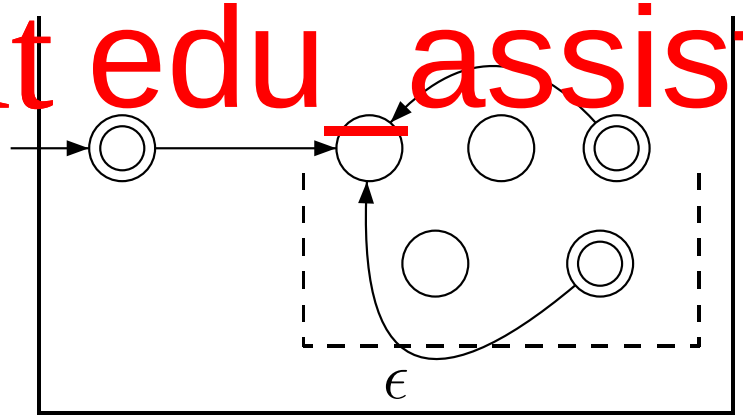
Closure Results for Regular Languages

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

Proof:

languag
on the righ

Here is how we construct an NFA to recognise A^*:



Closure Results for Regular Languages

Regular languages have several other closure properties.

They are c

- int
- complement, A^c
- difference (this follows, as $A \setminus B =$
- reversal.

Algorithms for Manipulating Automata

For some of these closure results, we will use the tutorials to develop useful DF

For this re

You will see, for example, how to systematically bui r languages $A \cap B$, out of DFAs for A an

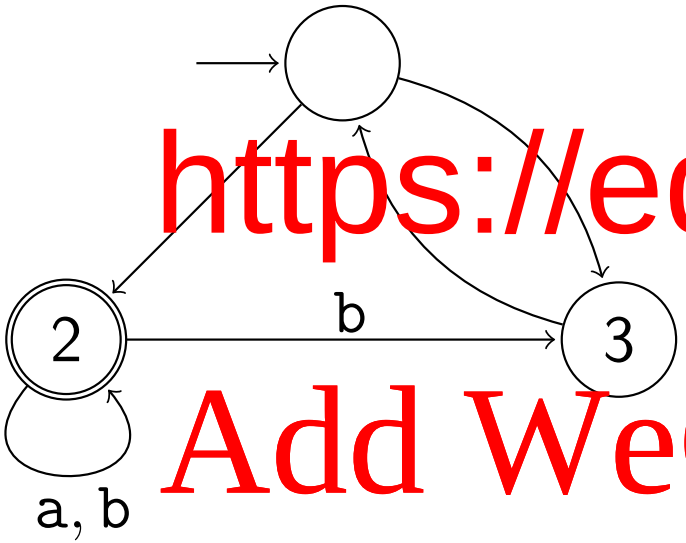
Equivalence of DFAs

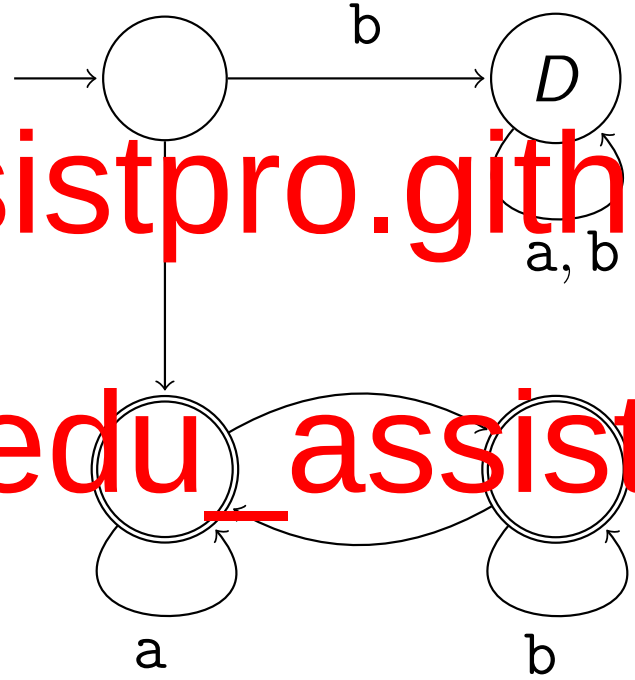
We can always find a minimal DFA for a given regular language (by mini ates).

Since a DF on is total and deterministic, we can test two DFAs for (modulo the names used for their states) by minimi

Minimizing DFAs

There is no guarantee that DFAs that are produced by the various algorithms, such as the subset construction method, will be minimal.





$A = \{1, 3\}$, $B^* = \{1, 2, 3\}$, $C^* = \{2, 3\}$, and $D = \emptyset$.

Generating a Minimal DFA

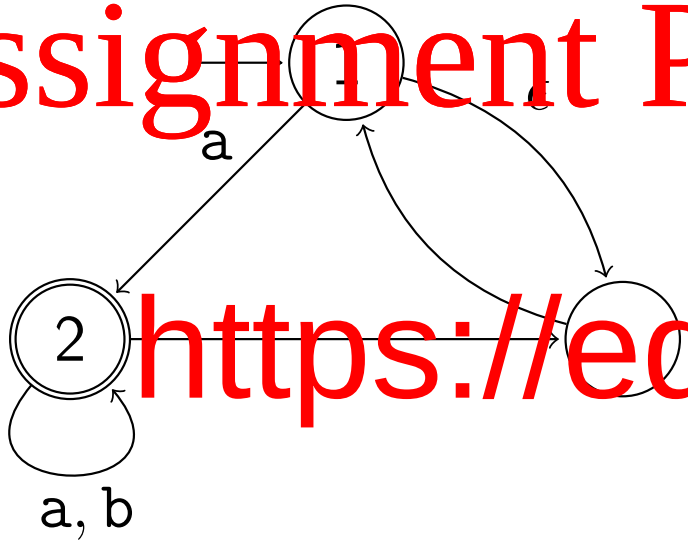
The following algorithm takes an NFA and produces an equivalent minimal DFA. Of course the input can also be a DFA.

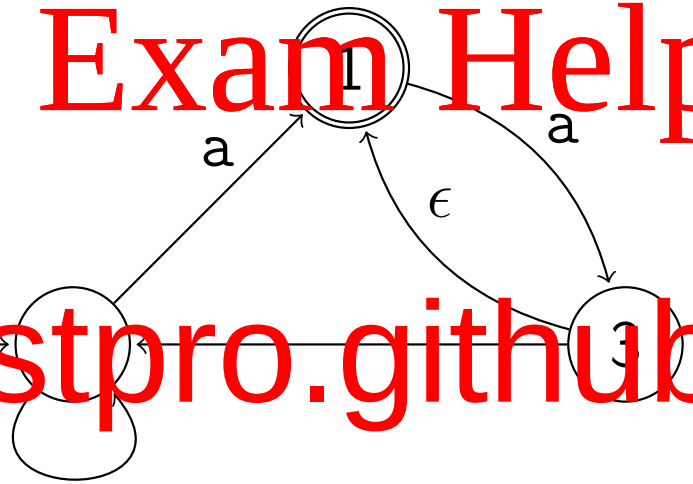
1. Re
2. De
3. Re
4. Determinize.

To reverse an NFA A with start states I $\neq \emptyset$: simply reverse every transition in A and swap I and F.

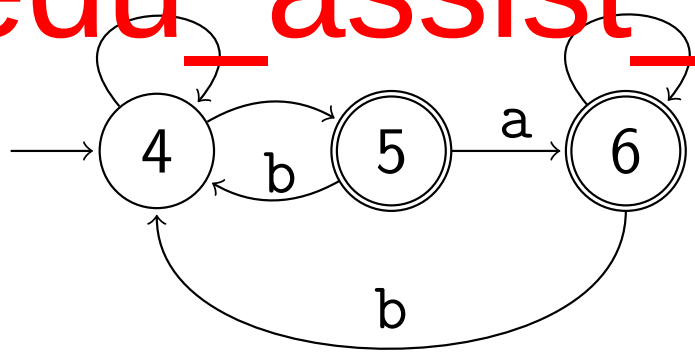
Minimization Example

Consider again the NFA that we determinized two slides ago.
Here it is on the left, with its reversal on the right:



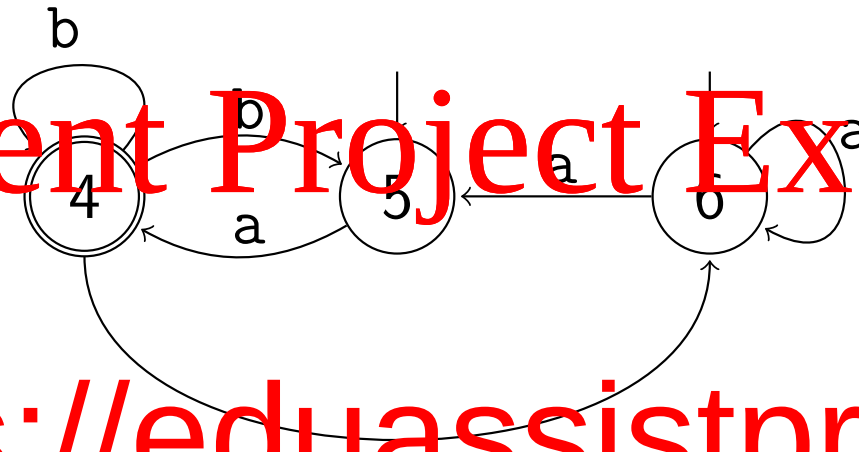


Now make the reversed NFA deterministic
(we have renamed the states to avoid later confusion:
4 corresponds to $\{2\}$, 5 to $\{1, 2\}$,
and 6 to $\{1, 2, 3\}$).

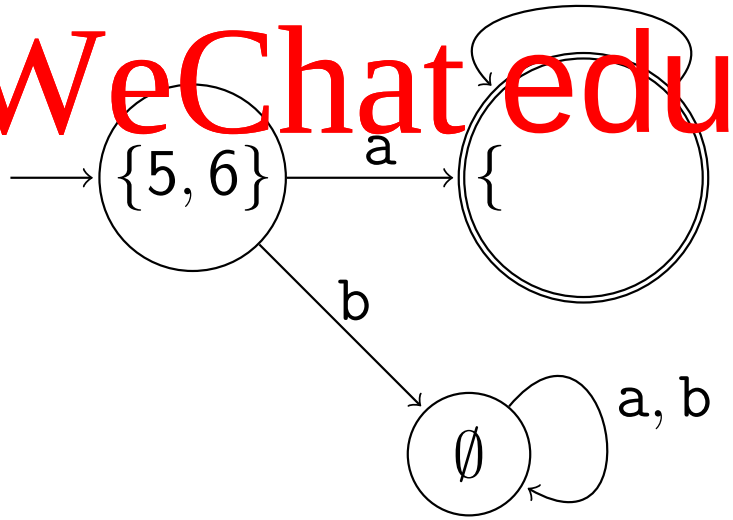


Minimization Example

Now reverse the result:



Finally m



Regular Expressions

Regular expressions is a notation for languages.

You are probably familiar with similar notation in Unix, Python or JavaScr ifferent things to d

Example:

$(0 \cup 1)(0 \cup 1)(0 \cup 1)((0 \cup 1)(0 \cup 1)(0$

non-empty strings with the engths that are multip

The star binds tighter than concatenation, which in turn binds tighter than union.

Regular Expressions

Syntax:

The regular expressions over an alphabet $\Sigma = \{a_1, \ldots, a_n\}$ are given by the grammar

$$regex \quad regexp^*$$

Semantics:

$$\begin{array}{lcl} L(a) & = & \{a \\ L(\epsilon) & = & \{\epsilon\} \\ L(\emptyset) & = & \emptyset \\ L(R_1 \cup R_2) & = & L(R_1) \cup L(R_2) \\ L(R_1\ R_2) & = & L(R_1) \circ L(R_2) \\ L(R^*) & = & L(R)^* \end{array}$$

Regular Expressions – Examples

ϵ : $\{\epsilon\}$

$(0 \cup \epsilon)(\epsilon \cup 1)$: $\{\epsilon, 0, 1$

1^* : all finite seque

$\epsilon \cup 1 \cup (\epsilon \cup 1)^*(\epsilon \cup 1)$: all finite sequence

$(1^*0^*)^*$: ?

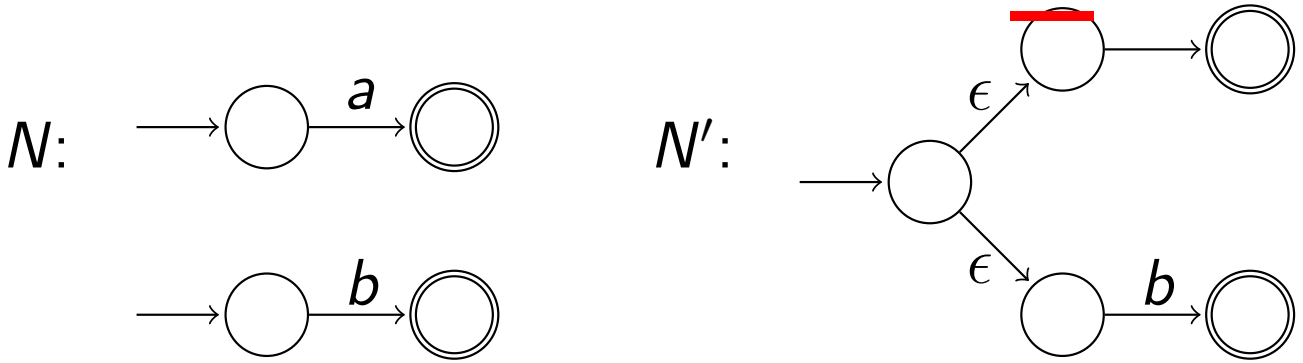
Regular Expressions vs Automata

Theorem: L is regular iff L can be described by a regular expression.

First note that, given NFA $N = (Q, \Sigma, \delta, I, F)$, we can build an equivalent NFA with at most one start state, like so: if $|I| \leq 1$, do nothing. Otherwise transform N to $N' = (Q \cup \{q_i\}, \Sigma, \delta', \{q_i\}, F)$ by adding a e in I:

$\delta(q, v)$ otherwise

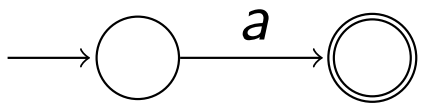
Example:



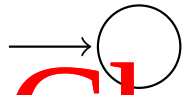
NFAs from Regular Expressions

We now show the 'if' direction of the theorem, by showing how to convert a regular expression R into an NFA that recognises $L(R)$.

The proof is by structural induction over the form of R.

Case R 

Case R 

Case $R = \emptyset$: Construct 

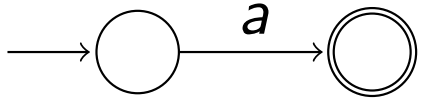
Case $R = R_1 \cup R_2$, $R = R_1\,R_2$, or $R =$

We already gave the constructions when we showed that regular languages are closed under the regular operations! They work because we can assume each NFA involved has a single start state.

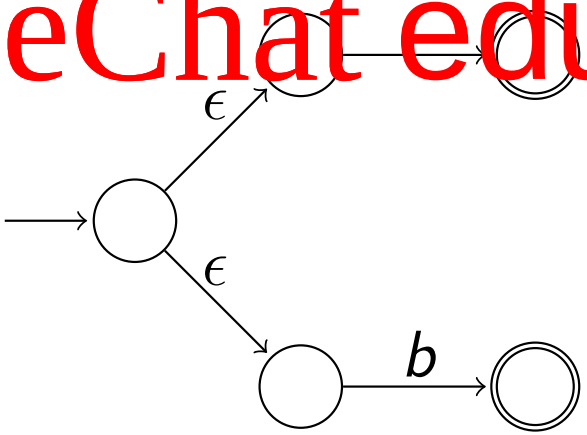
NFAs from Regular Expressions: Example

Let us construct, in the proposed systematic way, an NFA for $(a \cup b)^* bc$.

Start from innermost expressions and work out:

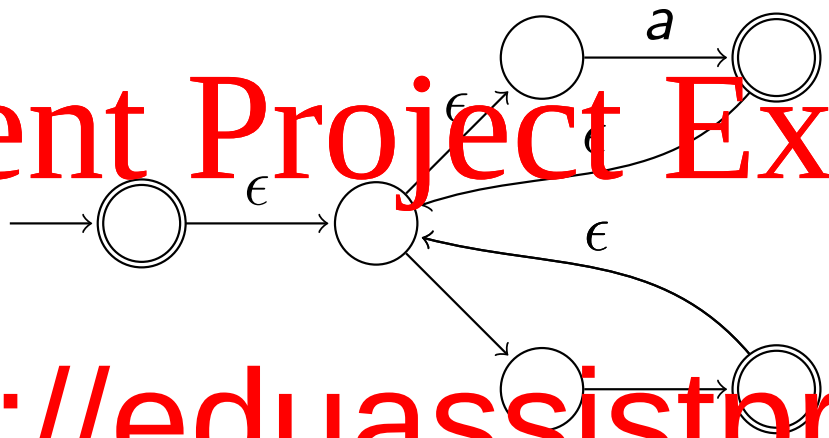


So a single-start state NFA for $a \cup b$ is:

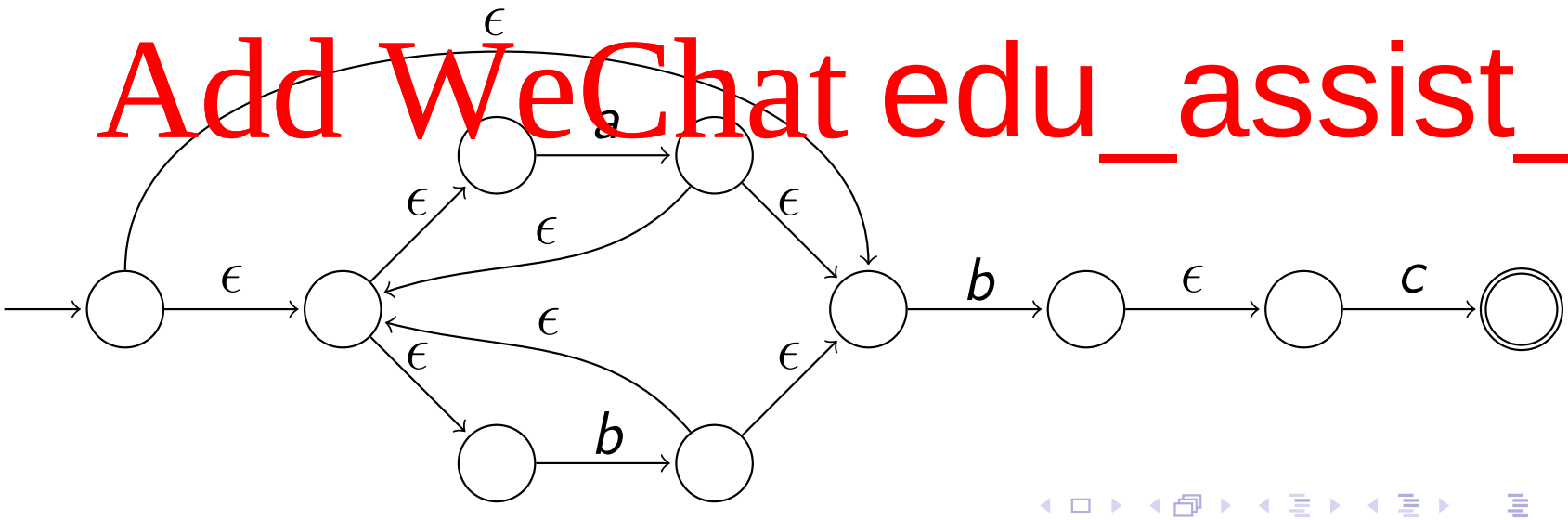


NFAs from Regular Expressions

Then $(a \cup b)^*$ yields:



Finally (

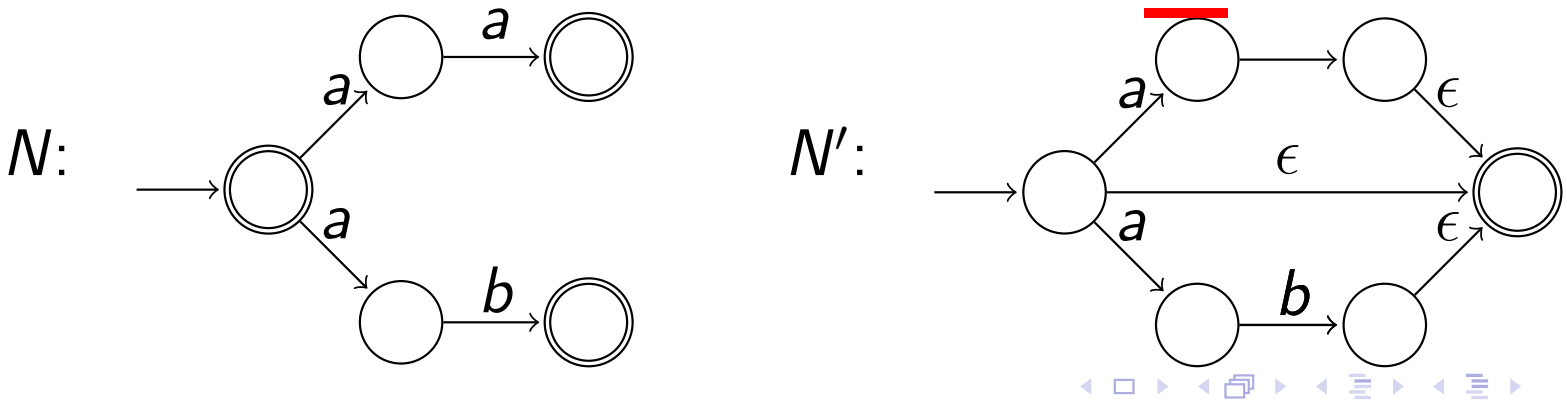


Regular Expressions from NFAs

We now show the 'only if' direction of the theorem.

First note that, given an NFA N, we can build an equivalent NFA with at most one accept state. We transform $N = (Q, \Sigma, \delta, I, F)$ to $N' = (Q \cup \{q_f\}, \Sigma, \delta', I, \{q_f\})$ like so: If $|F| \leq 1$, do nothing.
Otherwi te in F.
q_f beco

$$\delta'(q, v) = \begin{cases} \delta(q, v) & f \\ \text{otherwise} \end{cases}$$



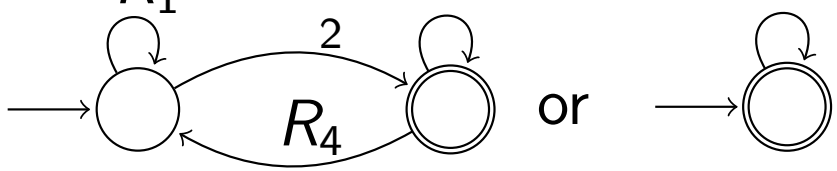
Regular Expressions from NFAs

We sketch how an NFA can be turned into a regular expression in a systematic process of "state elimination".

In the process, arcs get labelled with regular expressions.

Start by m gle start stat

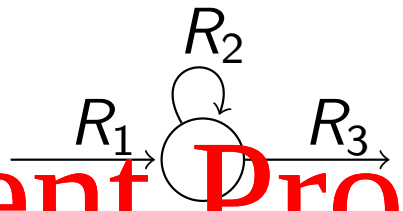
Repeatedly eliminate states that are neither start nor accept states.

The process produces either 

or

We get $(R_1 \cup R_2 R_3^* R_4)^* R_2 R_3^*$ in the first case; R^* in the second.

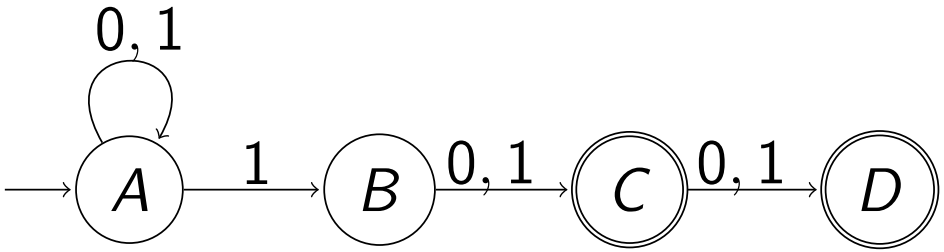
The State Elimination Process

Consider a node 

Any such pair of incoming/outgoing arcs get replaced by a single arc that by

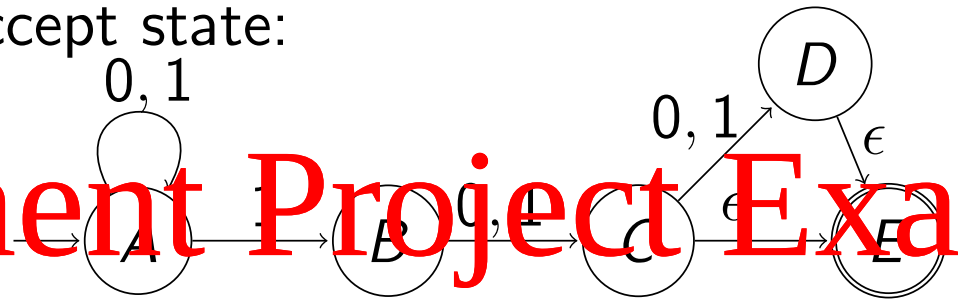
If there are ed
by $m \times n$ bypassing arcs when the node is remov

Let us illustrate the process on this example:

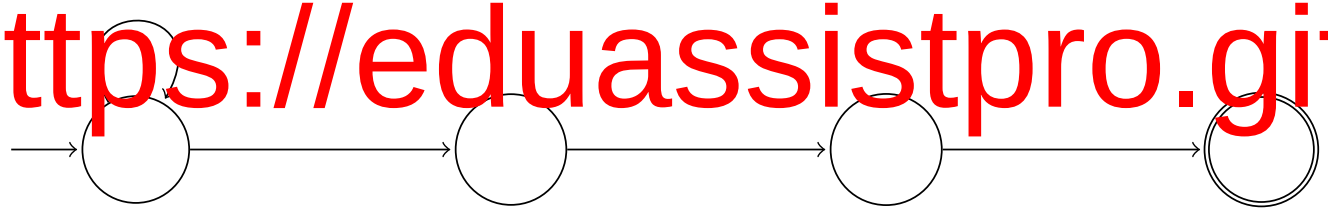


State Elimination Example

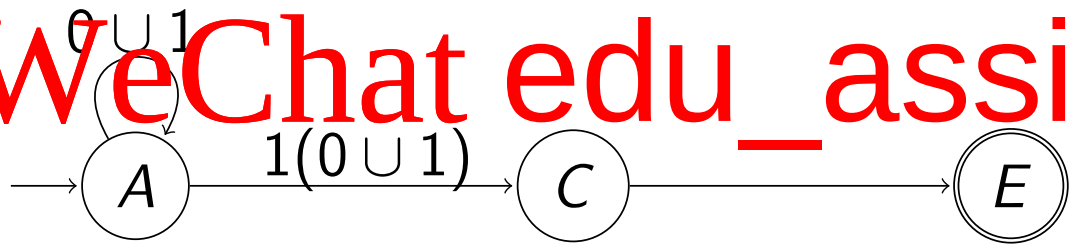
Create a single accept state:



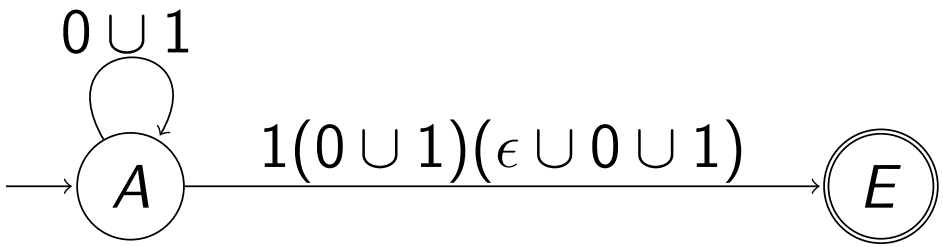
Elimina



Now eliminate B:

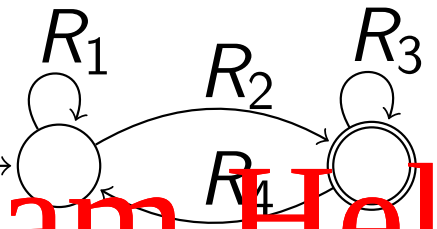


and then C:



State Elimination Example

Note that 

is 

with

- R_1
- R_2
- $R_3 = R_4 = \emptyset$

Hence the instance of the general "recipe" $(\quad)^*_3$ is

$$(0 \cup 1)^* 1 (0 \cup 1)(\epsilon \cup 0 \cup 1)$$

Sipser (see "Readings Online" on Canvas) provides more details of this kind of translation.

Some Useful Laws for Regular Expressions

$A \cup A = A$

$A \cup B = B \cup A$

$(A \cup B)$

$(A\ B)$

$\emptyset \cup A = A \cup \emptyset = A$

$\epsilon\ A = A\ \epsilon = A$

$\emptyset\ A = A\ \emptyset = \emptyset$

More Useful Laws for Regular Expressions

$(A \cup B)\,C = AC \cup BC$

$A\,(B \cup$

$(A^*)^* =$

$\emptyset^* = \epsilon^* = \epsilon$

$(\epsilon \cup A)^* =$

$(A \cup B)^* = (A^* B^*)^*$

Limitations of Finite-State Automata

Consider the language

Intuitive , because a DFA has n

Pumping lemma gives the formal proof.

Exercise: Is the language $L_1 = \{0^n 1^n \mid 0 \quad n \quad 999999999$ regular?

The Pumping Lemma for Regular Languages

This is the standard tool for proving languages non-regular.

Loosely, it says that if we have a regular language A and consider a sufficiently long string $s \in A$, then a recogniser for A must traverse some loop to accept s.

So A m of some sub

Pumping Lemma: If A is regular then the ch that for any string $s \in A$ with $|s| \geq p$, xyz, satisfying

1. $xy^i z \in A$ for all $i \geq 0$
2. $y \neq \epsilon$
3. $|xy| \leq p$

Proving the Pumping Lemma

Let DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognise A.

Let $p = |Q|$ and consider s with $|s| \geq p$. Let the number of states of M be p, let $|s| \geq p$.

In an accepting run for s,

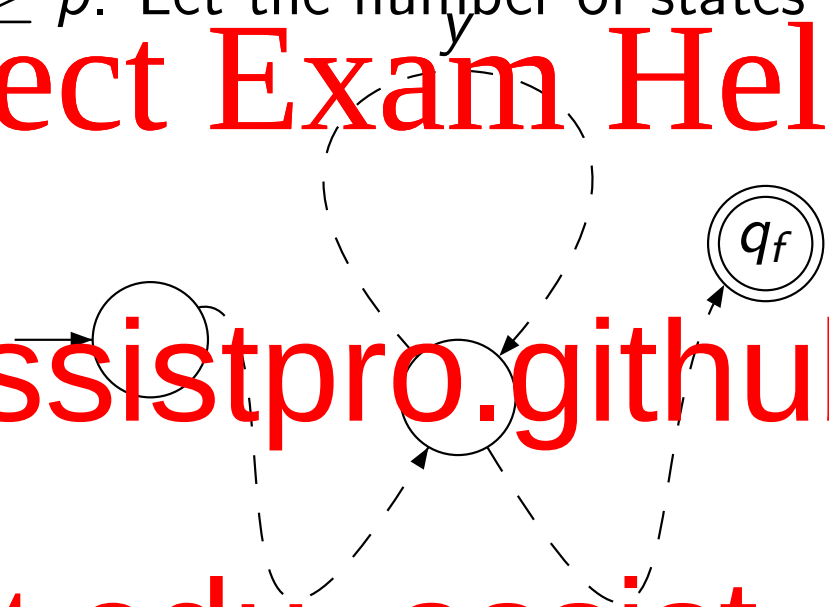
some stat

Let q_i b

At the first v

consumed, at the second, xy,

(strictly longer than x).



Consider the first time a state (q_i) is re-vi

of splitting s into x, y and z such that $xz, xyz, xyyz, \ldots$ are all in A.

Notice that $y \neq \epsilon$. Let $m + 1$ be the number of state visits when reading xy, then $|xy| = m \leq p$, because $m + 1$ is the number of state visits with only one repetition.

Using the Pumping Lemma

The pumping lemma says:

s can be written

We can us egular:

$$\forall p \exists s \in A : \begin{cases} s \text{ can't be written} \\ xyz \text{ such that } \ldots \end{cases}$$

Coming up with such an s is sometimes easy, sometimes difficult.

Pumping Example 1

We show that $B = \{0^n 1^n \mid n \geq 0\}$ is not regular.

Assume it is, and let p be the pumping length.

Conside

By the pu $i \geq 0$, $y \neq \epsilon$, an

Since $|xy| \leq p$, y consists entirely of 0s.

But then $xyyz \notin B$, a contradiction.

So we inevitably arrive at a contradiction if we assume that B is regular.

Pumping Example 2

$C = \{w$ r.

A simple p ld be regular, s r intersection.

Pumping Example 3

We show that $D = \{ww \mid w \in \{0,1\}^*\}$ is not regular.

Assume it is, and let p be the pumping length.

Conside

By the pu

ll $i \geq 0$,

$y \neq \epsilon$, and $|xy| \leq p$.

Since $|xy| \leq p$, y consists entirely of 0s.

But then $xyyz \notin D$, a contradiction.

Example 4 – Pumping Down

We show that $E = \{0^i 1^j \mid i > j\}$ is not regular.

Assume it is, and let p be the pumping length.

Conside

By the pu

ll $i \geq 0$,

$y \neq \epsilon$, and $|xy| \leq p$.

Since $|xy| \leq p$, y consists entirely of 0s.

But then $xz \notin E$, a contradiction.