# COMP30026 Models of Computation

## Regular Expressions and Non Regular Languages



Bach Le / Anna Kalenkova

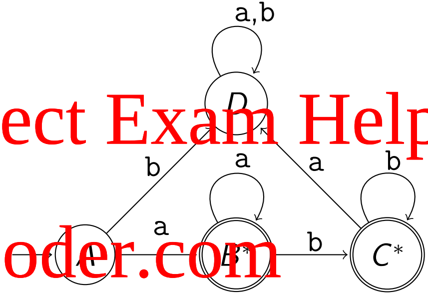


Lecture Week 8 Part 1

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 $\epsilon$-closure

| | a | b |
|---|---|---|
| A={1} | B* | D |
| B*={2,3} | B* | C* |
| C*={3} | D | C* |
| D=∅ | D | D |

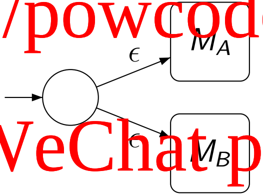
# Closure Results for Regular Languages

**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 recognisers. Together the two DFAs make up an NFA which recognises $A \cup B$:







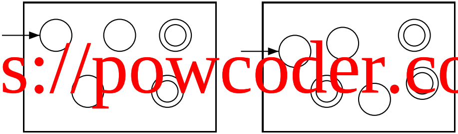

The $\epsilon$-transitions go to the start states of $M_A$ and $M_B$.

# Closure Results for Regular Languages

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



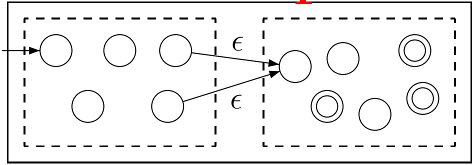
**Proof:** Let $A$ and $B$ be regular, with these DFAs as recognisers:





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





# That Last Construction, Formally



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

- $M_A = (Q, \Sigma, \delta, q_0, F)$
- $M_B = (Q', \Sigma, \delta', q'_0, F')$



A recogniser for $A \circ B$ is the NFA $(Q \cup Q', \Sigma, \delta'', \{q_0\}, F')$, where

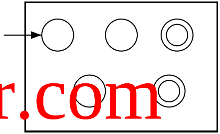


$$\delta''(q, v) = \begin{cases} \{\delta'(q, v)\} & \text{if } q \in Q' \text{ and } v \in \Sigma \\ \{\delta(q, v)\} & \text{if } q \in Q \text{ and } v \in \Sigma \\ \{q'_0\} & \text{if } q \in F \text{ and } v = \epsilon \end{cases}$$

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



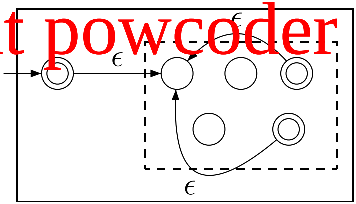
**Proof:** Let $A$ be a regular language with the DFA shown on the right as recogniser.







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



## Closure Results for Regular Languages



Regular languages have several other closure properties.

They are closed under



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

# Algorithms for Manipulating Automata



For some of these closure results, we will use the tutorials to develop useful DFA manipulation algorithms.



For this reason, the exercises are very important.

You will see, for example, how to systematically build DFAs for languages $A \cap B$, out of DFAs for $A$ and $B$.

# Equivalence of DFAs



We can always find a minimal DFA for a given regular language (by minimal we mean having the smallest possible number of states).

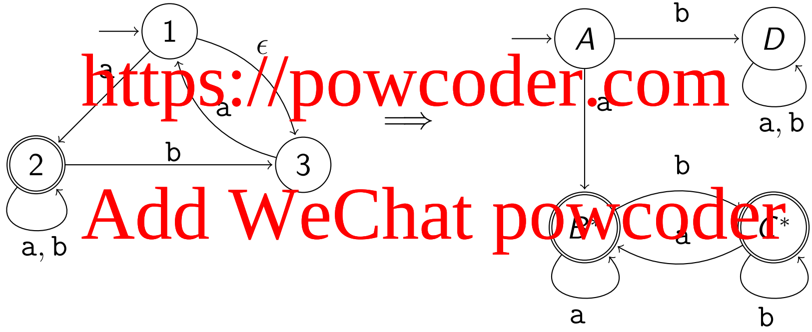


Since a DFA has a unique start state and the transition function is total and deterministic, we can test two DFAs for equivalence (modulo the names used for their states) by minimizing them.

# 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. Reverse the NFA;
2. Determinize the result;
3. Reverse again;
4. Determinize.



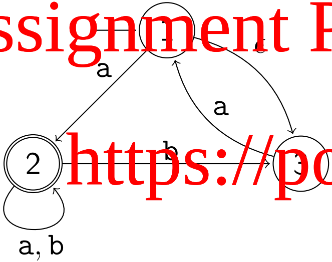


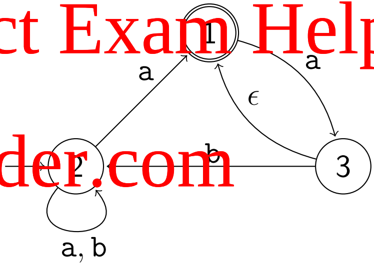
To reverse an NFA $A$ with start states $I$ and accept states $F$, $F \neq \emptyset$: simply reverse every transition in $A$ and swap $I$ and $F$.

# Minimization Example

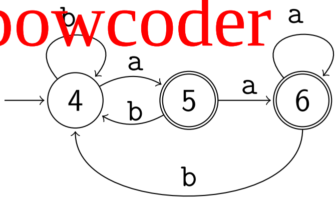
Consider again the NFA that we determinized two slides ago. Here it is on the left, with its reversal on the right:





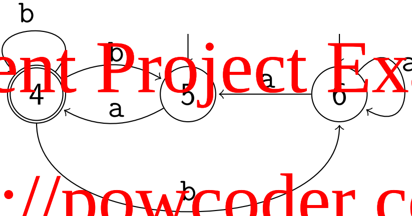




Now make the reversed NFA deterministic
(we have renamed the states to avoid later confusion:
4 corresponds to $\{2\}$, 5 to $\{1, 2\}$, and 6 to $\{1, 2, 3\}$).



# Minimization Example

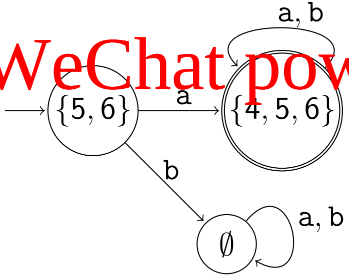
Now reverse the result:








Finally make the result deterministic:



# Regular Expressions

Regular expressions is a notation for languages.



You are probably familiar with similar notation in Unix, Python or JavaScript (but note also that "regular expression" means different things to different programmers).



**Example:**
$(0 \cup 1)(0 \cup 1)(0 \cup 1)((0 \cup 1)(0 \cup 1)(0 \cup 1))^*$ denotes the set of non-empty strings with the lengths that are multiple of 3.



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



$$\begin{array}{rcccccccccc} regexp & \rightarrow & a_1 & \mid & \cdots & \mid & a_n & \mid & \epsilon & \mid & \emptyset \\ & \mid & regexp \cup regexp & & & \mid & regexp\ regexp & & & \mid & regexp^* \end{array}$$



**Semantics:**

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

# Regular Expressions – Examples



$$
\begin{array}{rcl}
\epsilon & : & \{\epsilon\} \\
1 & : & \{1\} \\
110 & : & \{110\} \\
((0 \cup 1)(0 \cup 1))^* & : & \text{all binary strings of even length} \\
(0 \cup \epsilon)(\epsilon \cup 1) & : & \{\epsilon, 0, 1, 01\} \\
1^* & : & \text{all finite sequences of 1s} \\
\epsilon \cup 1 \cup (\epsilon \cup 1)^*(\epsilon \cup 1) & : & \text{all finite sequences of 1s} \\
(1^*0^*)^* & : & ?
\end{array}
$$

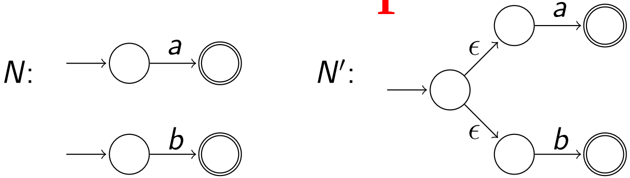
# Regular Expressions vs Automata

**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 new state $q_i$, with $\epsilon$ transitions from $q_i$ to each state in $I$:

$$\delta'(q, v) = \begin{cases} I & \text{if } q = q_i \text{ and } v = \epsilon \\ \delta(q, v) & \text{otherwise} \end{cases}$$

**Example:**

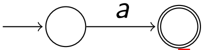


# NFAs from Regular Expressions

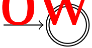
We now show the 'if' direction of the theorem, by showing how to convert a regular expression $R$ into an NFA that recognises $L(R)$.

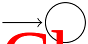


The proof is by structural induction over the form of $R$.

Case $R = a$: Construct 




Case $R = \epsilon$: Construct 

Case $R = \emptyset$: Construct 



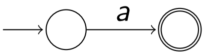
Case $R = R_1 \cup R_2$, $R = R_1\ R_2$, or $R = R_1^*$:
We already gave the constructions when we showed that regular languages are closed under the regular operations! They work because we can assume each NFA involved has a single start state.

# 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:

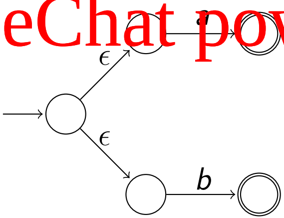




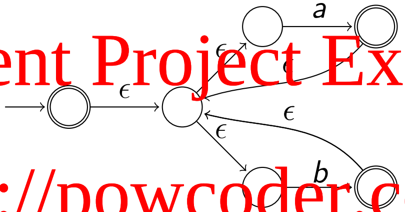


So a single-start state NFA for $a \cup b$ is:





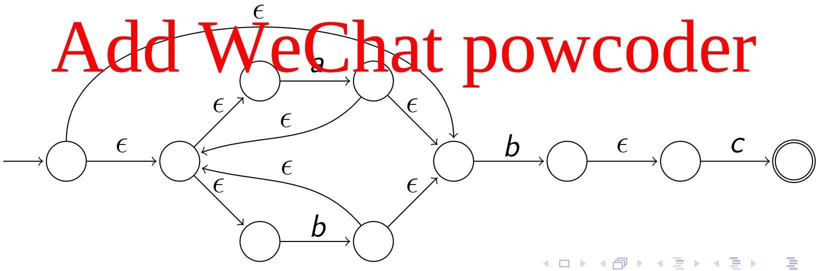
Then $(\mathtt{a} \cup \mathtt{b})^*$ yields:






Finally $(\mathtt{a} \cup \mathtt{b})^*\mathtt{bc}$ yields:





# Regular Expressions from NFAs

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



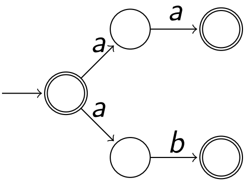
First note that, given an NFA $N$, we can build an equivalent NFA with at most one accept state. We transform $N = (Q, \Sigma, \delta, I, F)$ to $N' = (Q \cup \{q_f\}, \Sigma, \delta', I, \{q_f\})$ like so: If $|F| \leq 1$, do nothing. Otherwise add a new $q_f$ and $\epsilon$ transitions to $q_f$ from each state in $F$. $q_f$ becomes the only accept state:



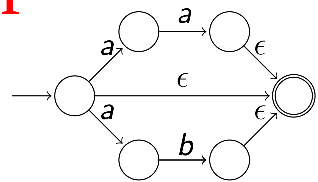
$$\delta'(q, v) = \begin{cases} \delta(q, v) \cup \{q_f\} & \text{if } q \in F \text{ and } v = \epsilon \\ \delta(q, v) & \text{otherwise} \end{cases}$$



$N$:




$N'$:



# 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.

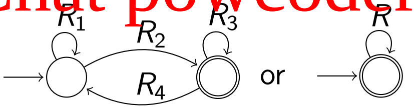
Start by making sure the NFA has a single accept state and a single start state.



Repeatedly eliminate states that are neither start nor accept states.

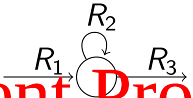


The process produces either




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 



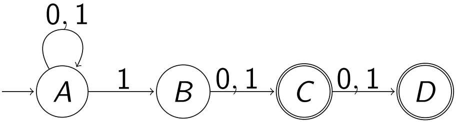

Any such pair of incoming/outgoing arcs get replaced by a single arc that bypasses the node. The new arc gets the label $R_1 R_2^* R_3$.



If there are $m$ incoming and $n$ outgoing arcs, these arcs are replaced by $m \times n$ bypassing arcs when the node is removed.

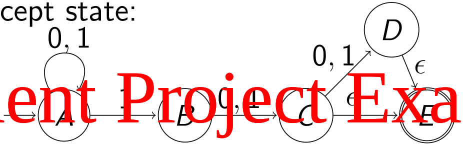
Let us illustrate the process on this example:





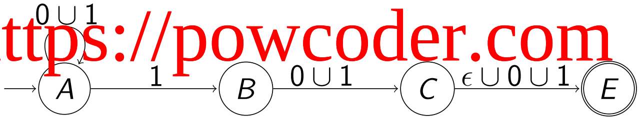
# State Elimination Example

Create a single accept state:





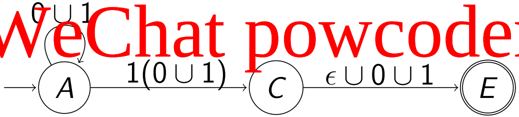

Eliminate $D$ (and use regular expressions with all arcs):



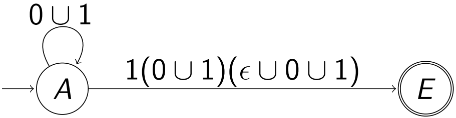




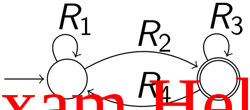

Now eliminate $B$:




and then $C$:



Note that 

is 




with



- $R_1 = 0 \cup 1$
- $R_2 = 1(0 \cup 1)(\epsilon \cup 0 \cup 1)$
- $R_3 = R_4 = \emptyset$



Hence the instance of the general "recipe" $(R_1 \cup R_2 R_3^* R_4)^* R_2 R_3^*$ 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

$$A \cup A = A$$



$$A \cup B = B \cup A$$

$$(A \cup B) \cup C = A \cup (B \cup C) = A \cup B \cup C$$



$$(A\ B)\ C = A\ (B\ C) = A\ B\ C$$

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



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

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

# More Useful Laws for Regular Expressions



$(A \cup B)\ C = A\ C \cup B\ C$

$A\ (B \cup C) = A\ B \cup A\ C$



$(A^*)^* = A^*$

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



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

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

# Limitations of Finite-State Automata



Consider the language

$$\{0^n 1^n \mid n \geq 0\} = \{\epsilon, 01, 0011, 000111, \ldots\}$$



Intuitively we cannot build a DFA to recognise this language, because a DFA has no memory of its actions so far.

Pumping lemma gives the formal proof.



**Exercise:** Is the language $L_1 = \{0^n 1^n \mid 0 \leq n \leq 999999999\}$ 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 $A$ must contain infinitely many strings exhibiting repetition of some substring in $s$.



**Pumping Lemma:** If $A$ is regular then there is a number $p$ such that for any string $s \in A$ with $|s| \geq p$, $s$ can be written as $s = xyz$, satisfying



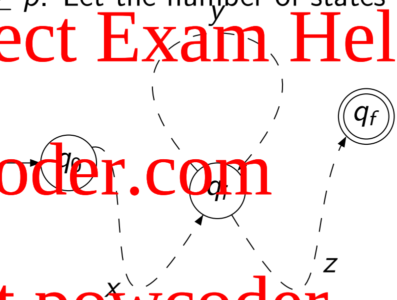
1. $xy^iz \in A$ for all $i \geq 0$
2. $y \neq \epsilon$
3. $|xy| \leq p$

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 state must be re-visited. Let $q_i$ be the first such state. At the first visit, $x$ has been consumed, at the second, $xy$, (strictly longer than $x$).








Consider the first time a state ($q_i$) is re-visited. This suggests a way 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:

$$A \text{ regular} \Rightarrow \exists p \forall s \in A : \left\{ \begin{array}{l} s \text{ can be written} \\ xyz \text{ such that } \ldots \end{array} \right.$$



We can use its contrapositive to show that a language is non-regular:

$$\forall p \exists s \in A : \left\{ \begin{array}{l} s \text{ can't be written} \\ xyz \text{ such that } \ldots \end{array} \right\} \Rightarrow A \text{ not regular}$$



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

# Pumping Example 1

We show that $B = \{0^n1^n \mid n \geq 0\}$ is not regular.



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

Consider $0^p1^p \in B$ with length greater than $p$.



By the pumping lemma, $0^p1^p = xyz$, with $xy^iz$ in $B$ for all $i \geq 0$, $y \neq \epsilon$, and $|xy| \leq p$.

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



$C = \{w \mid w$ has an equal number of 0s and 1s$\}$ is not regular.



A simple proof: If $C$ were regular then also $B$ from before would be regular, since $B = C \cap 0^*1^*$ and regular languages are closed under 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.



Consider $0^p10^p1 \in D$ with length greater than $p$.

By the pumping lemma, $0^p10^p1 = xyz$, with $xy^iz$ in $D$ for all $i \geq 0$, $y \neq \epsilon$, 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.

Consider $0^{p+1}1^p \in E$.



By the pumping lemma, $0^{p+1}1^p = xyz$, with $xy^iz$ in $E$ for all $i \geq 0$, $y \neq \epsilon$, and $|xy| \leq p$.



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

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