# COMP30026 Models of Computation

## Regular Expressions and Non-Regular Languages



Anna Kalenkova



Lecture Week 8 Part 1

Semester 2, 2020

# 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



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



**Semantics:**

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

# Regular Expressions – Examples



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

# Regular Expressions vs Automata

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

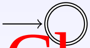
Let us first show the 'if' direction, by showing how to convert a regular expression $R$ into an NFA that recognises $L(R)$.



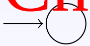
The proof is by structural induction over the form of $R$.



Case $R = a$: Construct 


Case $R = \epsilon$: Construct 



Case $R = \emptyset$: Construct 

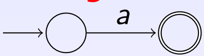
Case $R = R_1 \cup R_2$, $R = R_1\ R_2$, or $R = R_1^*$:
We already gave the constructions when we showed that regular languages were closed under the regular operations.

# NFAs from Regular Expressions

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



Start from innermost expressions and work out:

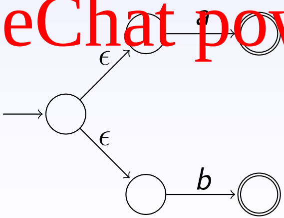




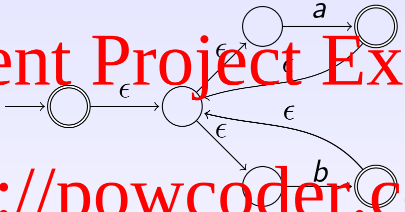


So $a \cup b$ yields:



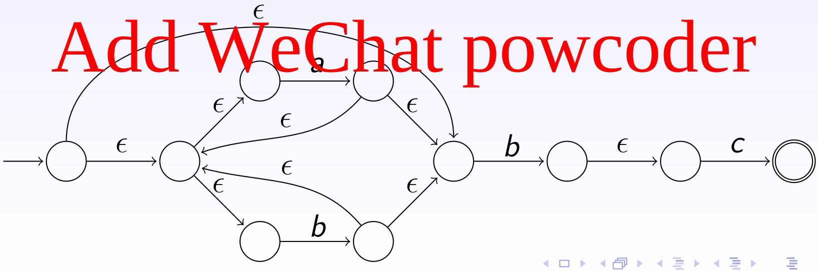


Then $(\mathtt{a} \cup \mathtt{b})^*$ yields:






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



# Regular Expressions from NFAs

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

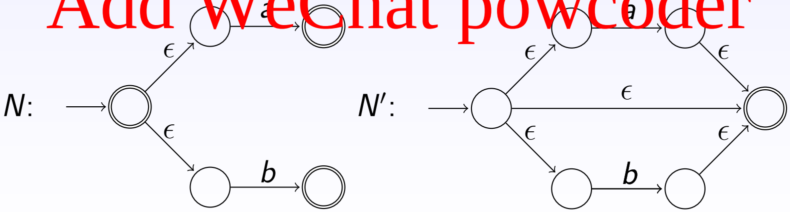


Note that, given an NFA $N$, we can easily build an equivalent NFA with at most one accept state. We transform $N = (Q, \Sigma, \delta, q_0, F)$ to $N' = (Q \cup \{q_f\}, \Sigma, \delta', q_0, \{q_f\})$ by adding a new $q_f$, with $\epsilon$ transitions to $q_f$ from each state in $F$. $q_f$ becomes the only accept state:



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





# Regular Expressions from NFAs

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



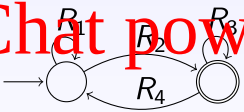
In the process, arcs get labelled with regular expressions.

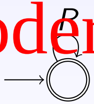
Start by making sure the NFA has a single accept state.



Repeatedly eliminate states that are neither start nor accept states.



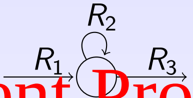
The process produces either 

or 


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

Note that some $R$s may be $\epsilon$ or $\emptyset$.

Consider a node





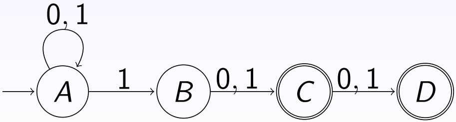

Any such pair of incoming/outgoing arcs get replaced by a single arc that bypasses the node. The new arc gets the label $R_1 R_2^* R_3$.



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

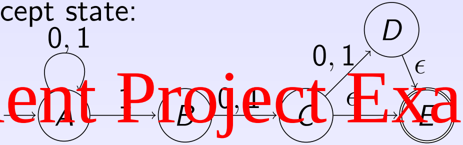
Let us illustrate the process on this example:



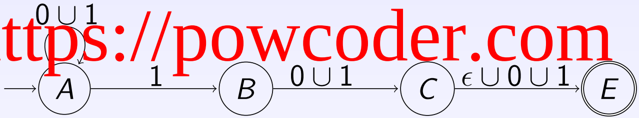


# State Elimination Example

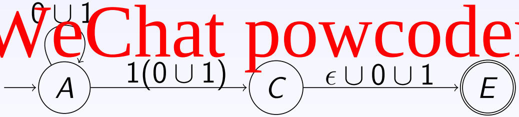
Create a single accept state:



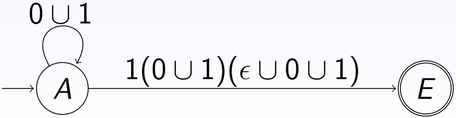

Eliminate $D$ (and use regular expressions with all arcs):




Now eliminate $B$:

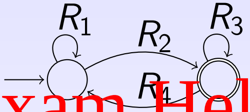



and then $C$:



# State Elimination Example

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.

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

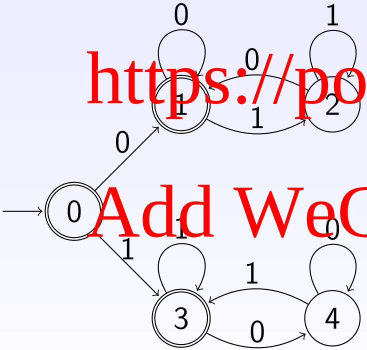


What about $L_2 = \left\{ w \;\middle|\; \begin{array}{l} w \text{ has an equal number of occurrences} \\ \text{of the substrings 01 and 10} \end{array} \right\}$ ?

# Language $L_2$

$$L_2 = \left\{ w \;\middle|\; \begin{array}{l} w \text{ has an equal number of occurrences} \\ \text{of the substrings 01 and 10} \end{array} \right\} ?$$





# 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. $|xy| \leq p$

We call $p$ the pumping length.

# Proving the Pumping Lemma

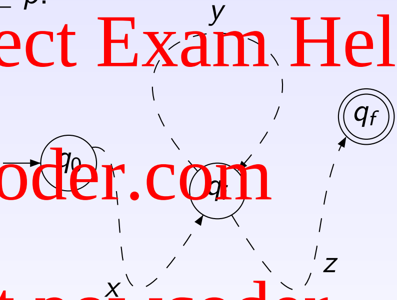
Let DFA $M = (Q, \Sigma, \delta, q_0, F)$ recognise $A$.

Let $p = |Q|$ and consider $s$ with $|s| \geq p$.

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




Notice that $y \neq \epsilon$. Also, if input consumed has length $k$ then the number of state visits is $k+1$. Let $m+1$ be the number of state visits when reading $xy$, then $|xy| = m \leq p$. Notice that $m \leq p$, 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 $i \geq 0$.

But $y$ cannot consist of all 0s, since $xyyz$ then has more 0s than 1s.



Similarly $y$ cannot consist of all 1s. And if $y$ has at least one 0 and one 1, then some 1 comes before some 0 in $xyyz$.

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.



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

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



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

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

# Pumping Example 2

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



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

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



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

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



---

A simpler alternative 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.