Monash University
Faculty of Information Technology

FIT2014 Theory of Computation

Lecture 9

Kleene's Theorem. II.

FA $\longrightarrow$ Regexp

slides by Graham Farr

Overview

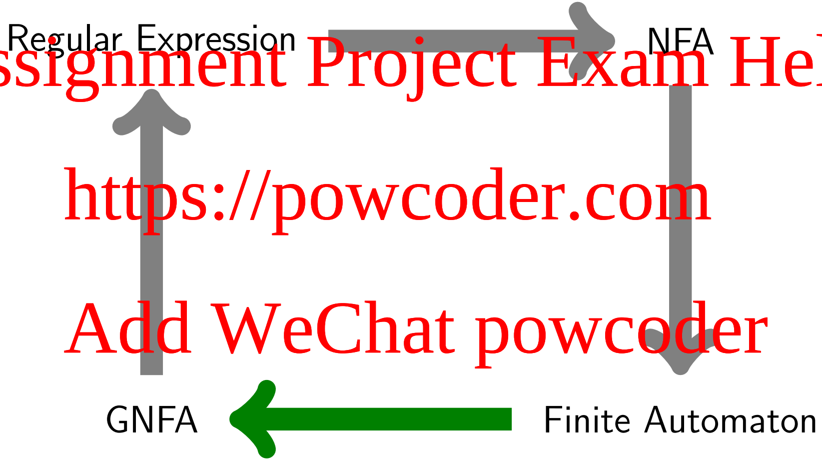
Previous lecture:

- Convert Regular Expressions to NFA
- Convert NFA to FA

Today:

- Generalised Nondeterministic Finite Automata
- Convert FA to GNFA
- Convert GNFA to Regular Expression

Kleene's Theorem



Generalised Nondeterministic Finite Automaton (GNFA)

Definitions

A **Generalised Nondeterministic Finite Automaton (GNFA)** is defined as for an NFA except that transitions may be labelled by regular expressions, not just by single letters.

A string w is **accepted** by a given GNFA if it can be divided into substrings, $w = w_1 \cdots w_k$, such that there is some sequence of transitions, starting at the Start State, finishing at the Final State, and labelled by regular expressions $R_1, \ldots, R_k$, such that, for all i, w_i matches R_i.

If a string w is not accepted by the GNFA, then it is **rejected**.

Standard GNFA

A **standard GNFA** is a GNFA in which:

- there is just one Final State, and it is not the Start State;
- the Start State has no incoming transitions and the Final State has no outgoing transitions.

Notes:

- Sometimes (e.g., in Sipser), standard GNFAs are required to have arcs going between every pair of states, subject to the prohibition on incoming arcs into the Start State and outgoing arcs from the Final State. If a transition actually cannot occur between two states, then the arc between them is labelled by the regular expression $\emptyset$, which prevents the transition from being used.
- This is not really needed, for the algorithm we will describe. But it may make proving properties of that algorithm easier.
- Standard GNFAs are sometimes called GNFAs of "special form" in Sipser.

From FA to Standard GNFA

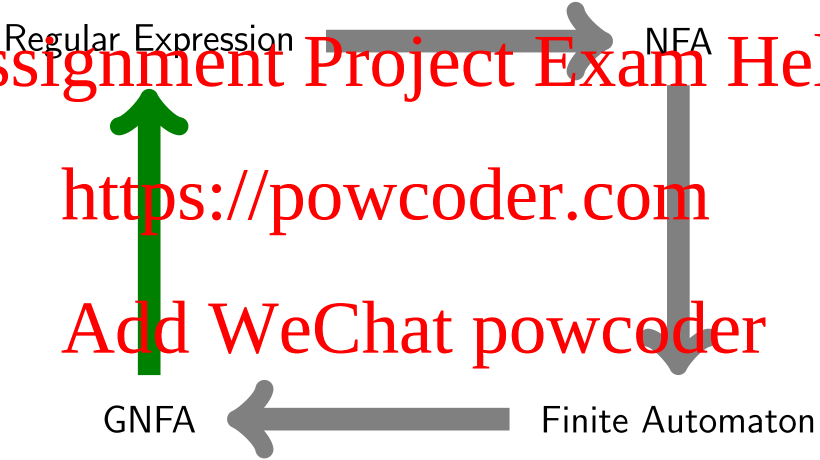
Given a FA (or an NFA):

1. Ensure there is a single Final State, with incoming arcs only.
   - If necessary: add new Final State, add new transitions labelled ε from the previous Final States to this new one, and make those states no longer Final.
2. Ensure there is a single Start State, with outgoing arcs only.
   - If necessary: add a new Start State, add new transitions labelled ε from this new Start State to the previous Start States, and make those states no longer Start states.

The letters on the arcs of the FA/NFA are already regular expressions in their own right.

The GNFA constructed by this process accepts the same language as the original FA.

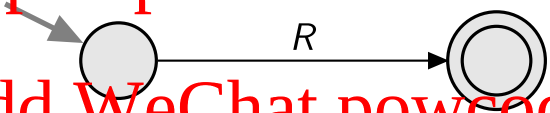
Kleene's Theorem



From Standard GNFA to Regular Expression

Starting with a Standard GNFA, we convert it to an equivalent GNFA with one fewer state.

We keep doing this until we have a GNFA with just a Start State, a Final State, and one transition:



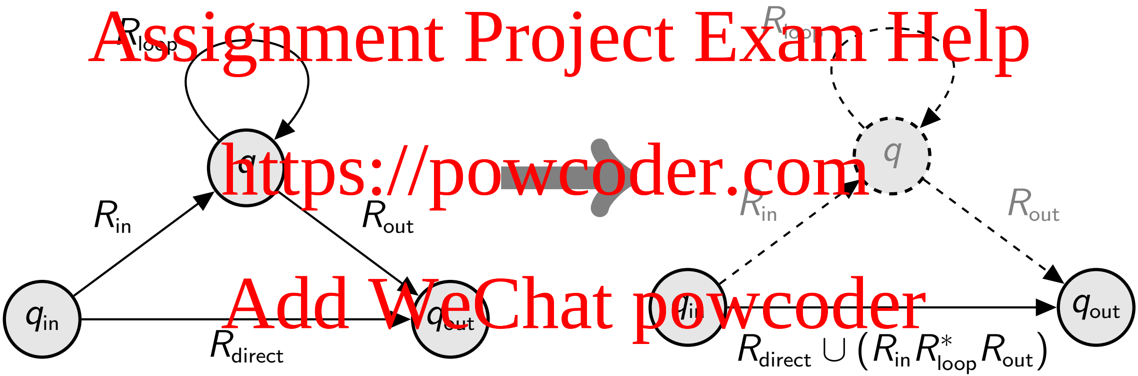
The regular expression R on this transition defines the same language as the original GNFA.

From GNFA to Regular Expression

Notation:

q	some non-Start, non-Final state.
q_{in}	any non-Final state.
R_{in}	the regular expression on the transition from q_{in} to q.
q_{out}	any non-Start state.
R_{out}	the regular expression on the transition from q to q_{out}.
R_{loop}	the regular expression on the transition from q to itself.
R_{direct}	the regular expression on the transition from q_{in} to q_{out}.

From GNFA to Regular Expression



From GNFA to Regular Expression

Ensure this replacement is done for all q_{in}, q_{out}.

Then q is removed, and we have an equivalent GNFA to the one we started with.

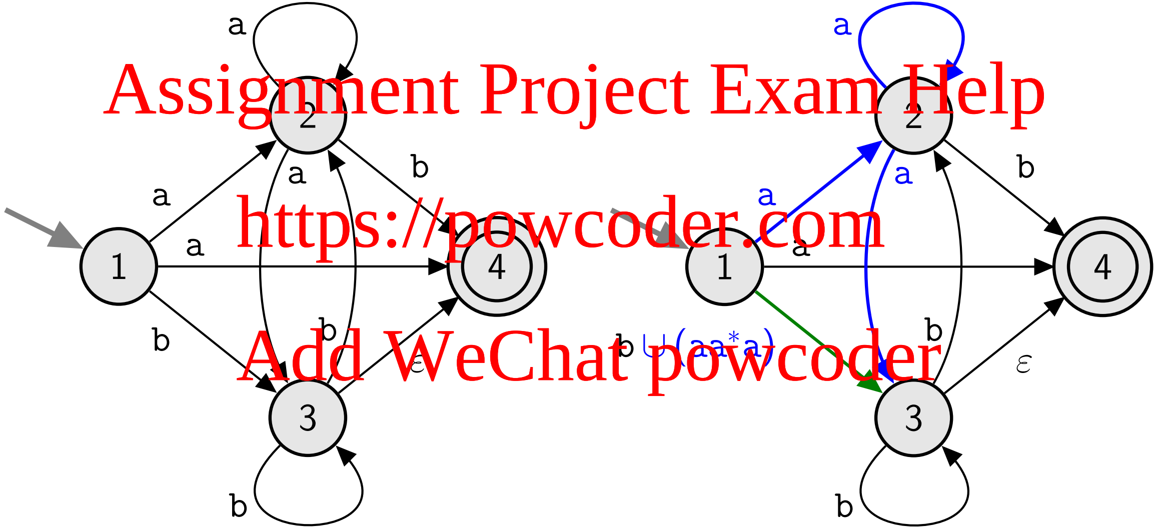
Keep doing this whole procedure, removing one state at a time, until you are left with just the Start State and the Final State, with a single transition between them.

The regular expression on this transition is the one you want.

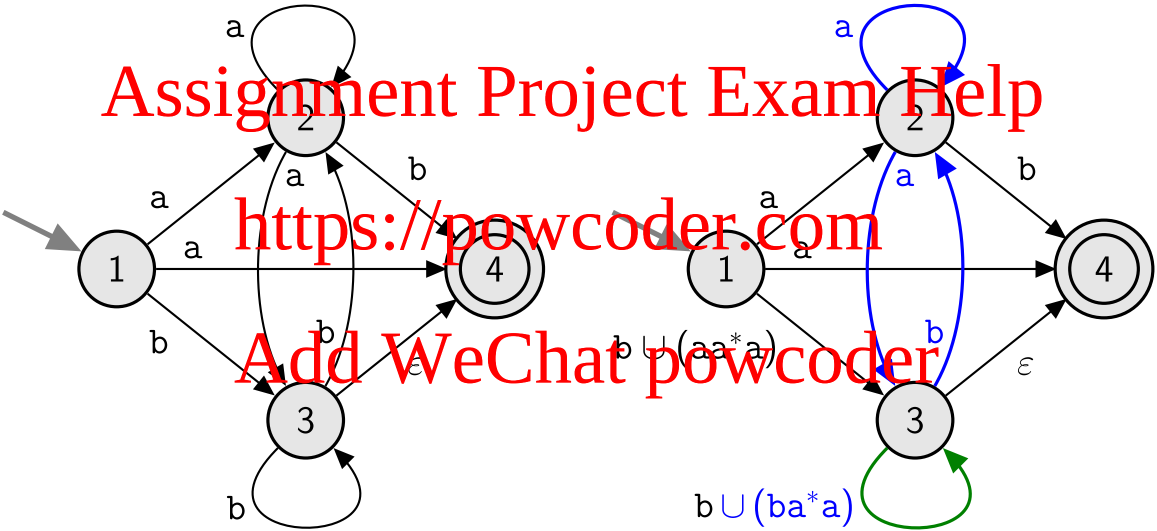
It matches precisely those strings accepted by the original GNFA.

Examples: Sipser, pp. 75–76.

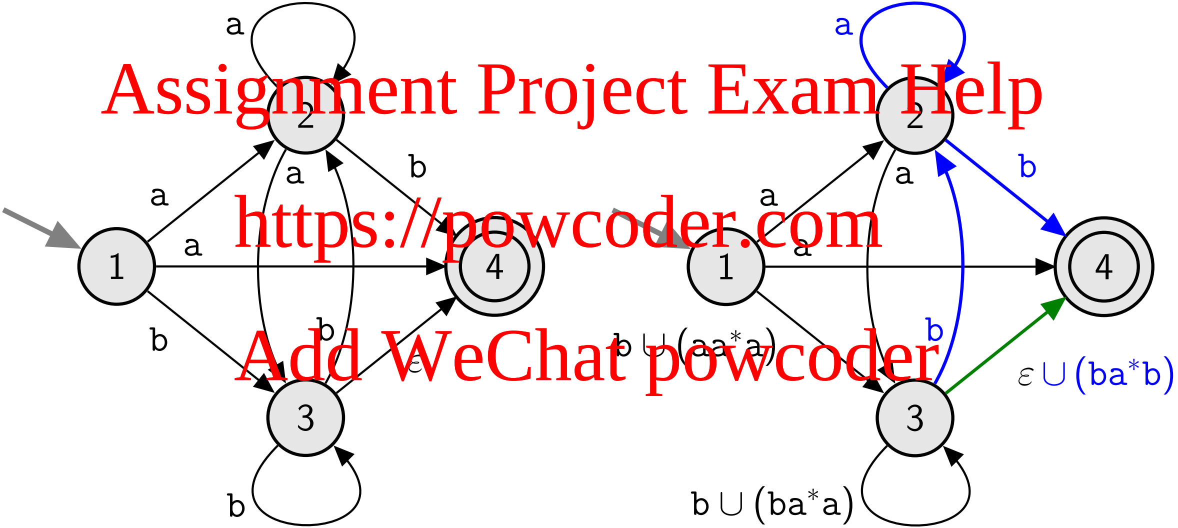
From GNFA to Regular Expression



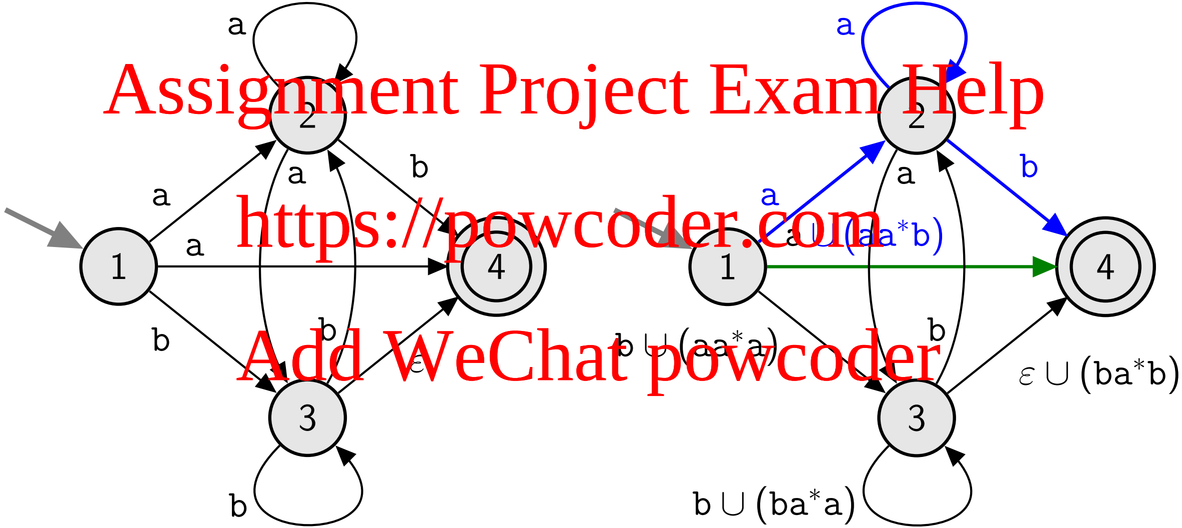
From GNFA to Regular Expression



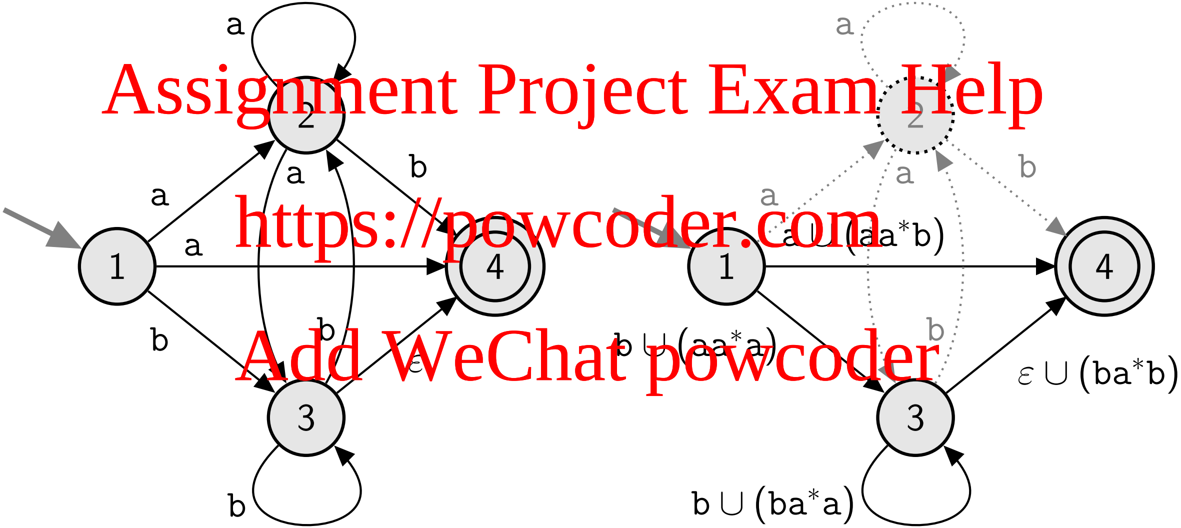
From GNFA to Regular Expression



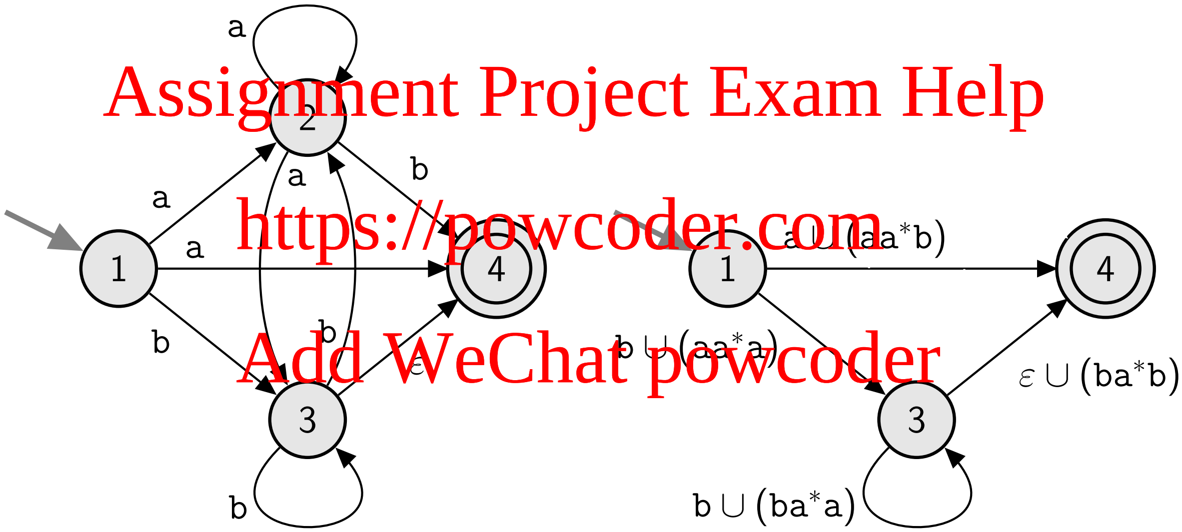
From GNFA to Regular Expression



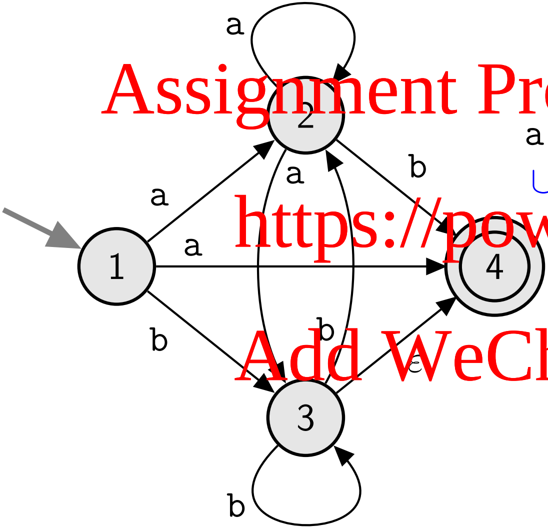
From GNFA to Regular Expression

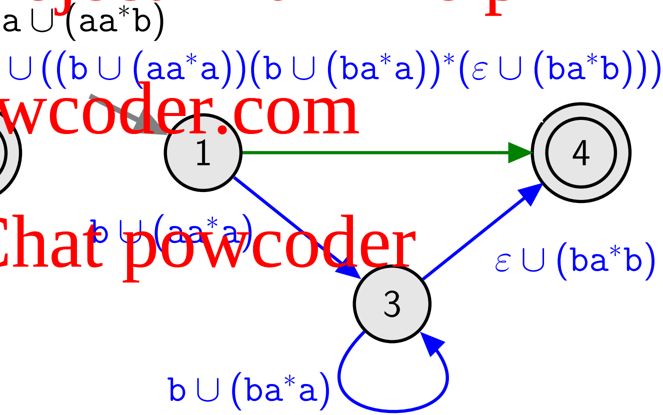


From GNFA to Regular Expression

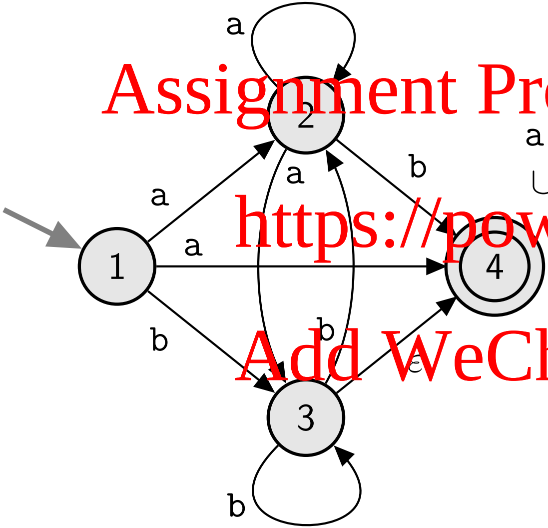


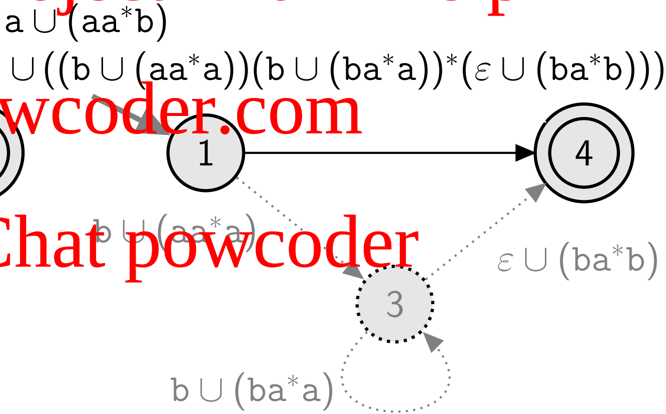
From GNFA to Regular Expression



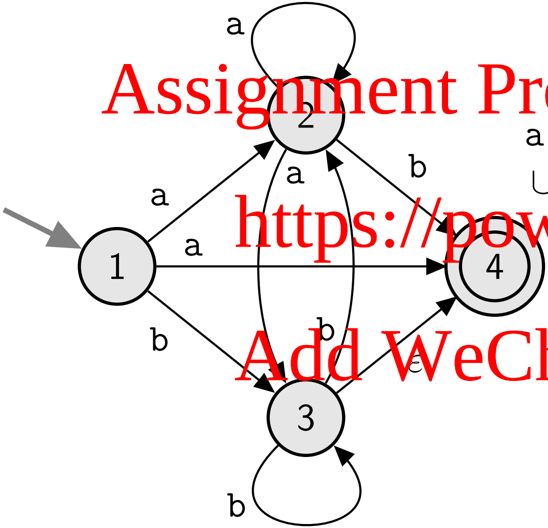


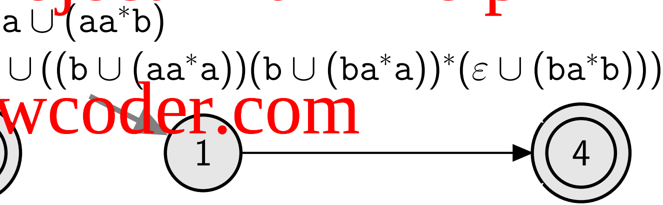
From GNFA to Regular Expression





From GNFA to Regular Expression





Complexity?

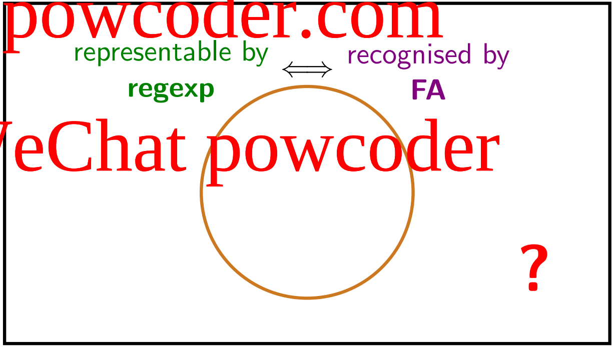
For FIT2004 students

Compare this algorithm with the Floyd-Warshall algorithm for the All Pairs Shortest Path problem.

Questions

- Can every language which is represented by a **regular expression** be described by a **finite automaton**? YES
- Can every language which is described by a **finite automaton** be represented by a **regular expression**? YES
- Can every language be represented by a **regular expression** or a **finite automaton**?

{ all languages }



Revision

Previous lecture:

- Understand Kleene's Theorem
- Be able to convert Regular Expressions into NFA
- Be able to convert NFA into a Finite Automaton

Today:

- Be able to convert a FA into a Regular Expression

Reference

Sipser, Ch. 1, especially pp. 66, 69–76.