

Lecture 7

Kleene's Theorem.

Slides by David Albrecht (2011) and Graham Farr (2013).

FIT2014 Theory of Computation

Overview

- Questions
- Kleene's Theorem
- Convert Regular Expressions to NFA
- Convert NFA to FA
- Convert FA to Regular Expression



Stephen Cole Kleene
(1909-1994)

<http://www-history.mcs.st-and.ac.uk/Biographies/Kleene.html>

Questions

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

Kleene's Theorem

Any language which can be defined by

- **Regular Expressions**
- **Finite Automata**
- **Nondeterministic Finite Automata (NFA)**
- **Generalized Nondeterministic Finite Automata (GNFA)**

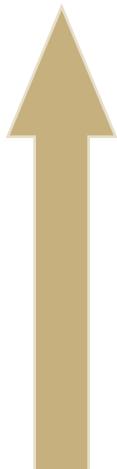
can be defined by any of the other methods.

Kleene's Theorem

Regular Expression



NFA



GNFA



Finite Automaton

Converting Regular Expression to NFA

Start with the graph.

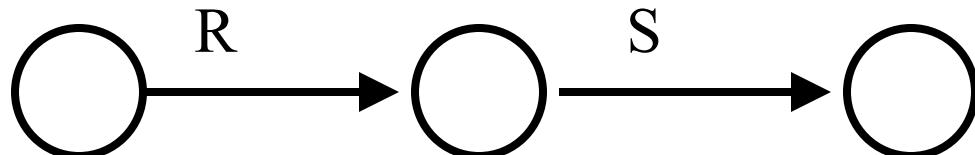


Apply the following rules until all edges are labelled with a letter or ϵ .

1. Delete any edge labelled with ϕ .
2. Transform any edge like



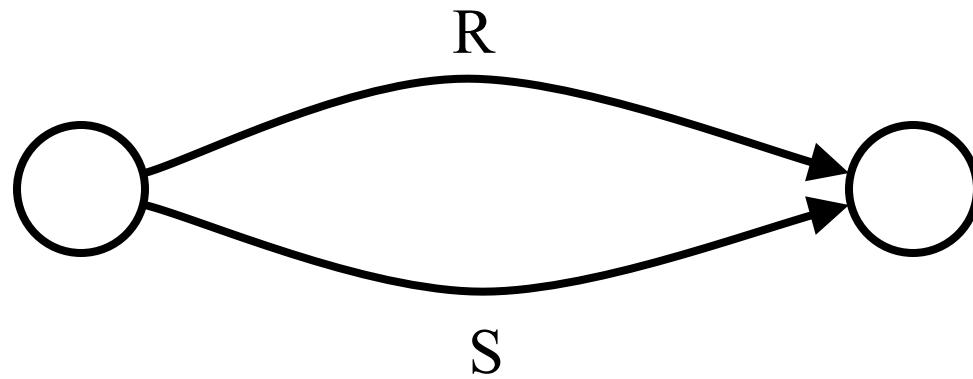
into



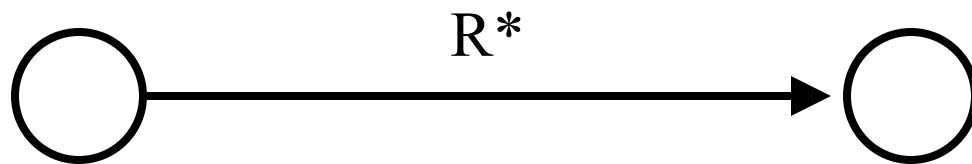
3. Transform any edge like:



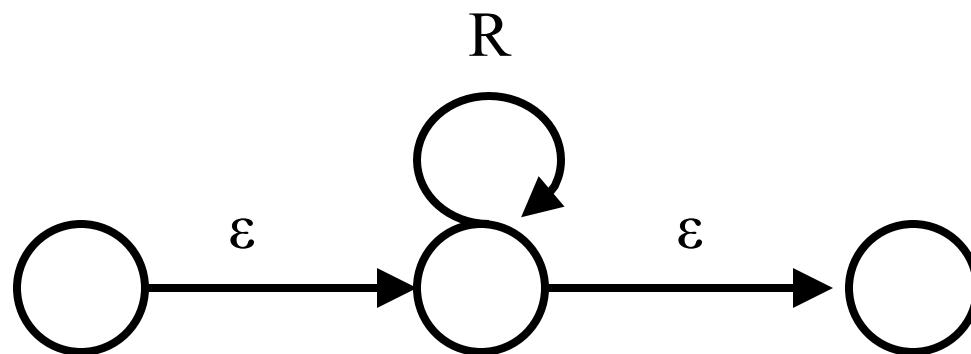
into

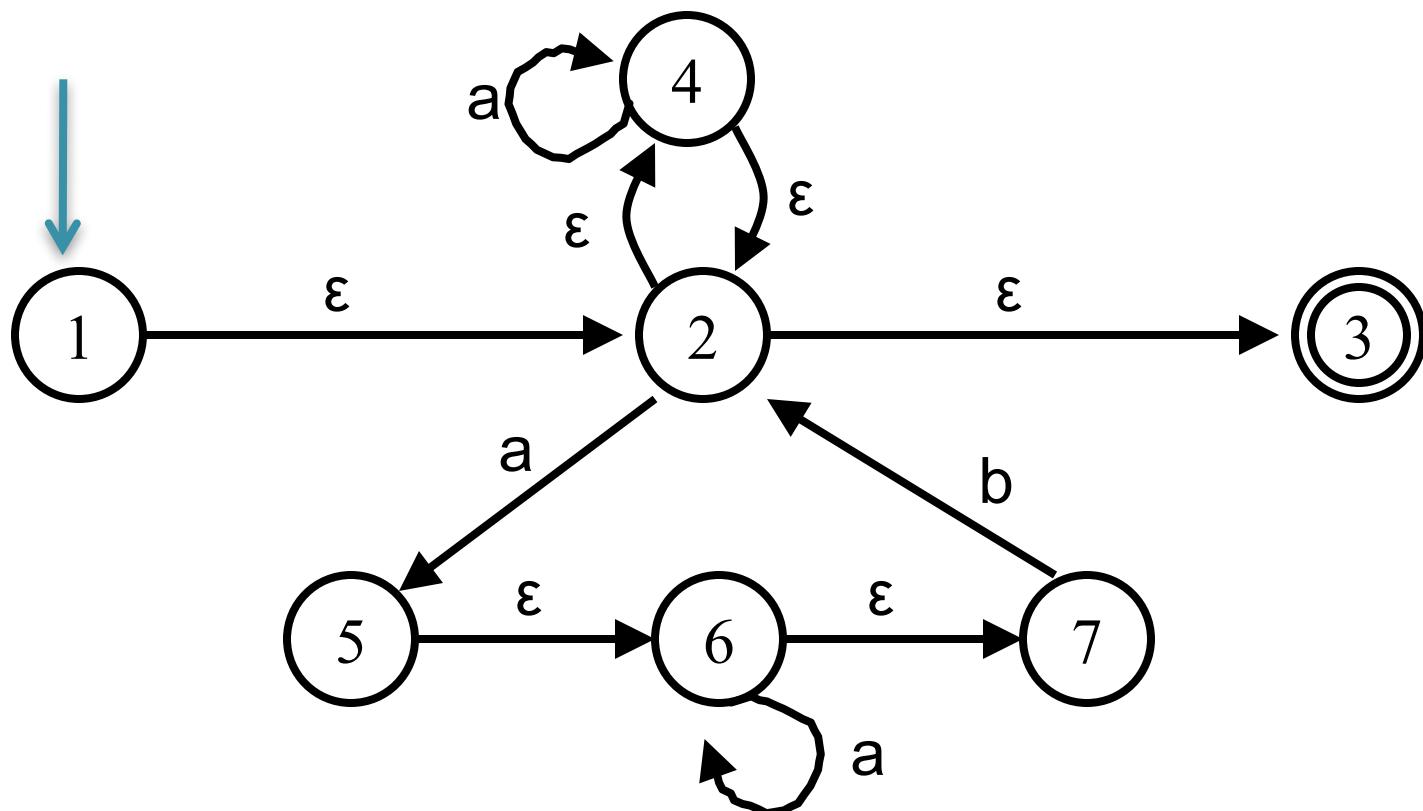


4. Transform any edge like:



into



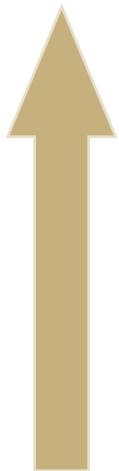
$$(a^* \cup aa^*b)^*$$


Kleene's Theorem

Regular Expression



NFA



GNFA



Finite Automaton

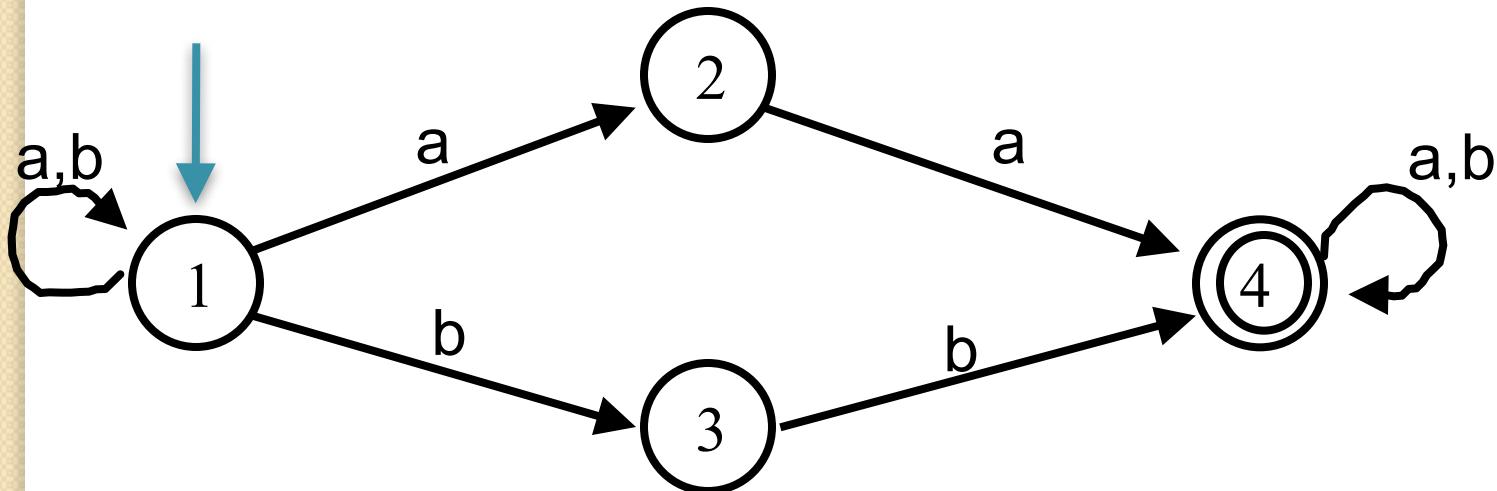
Converting a NFA to a FA

In a FA:

- Any string w traces a ***unique path***, starting from the Start State and ending at ***some unique state***, which we'll call $\text{endState}(w)$.
- The string w is accepted if $\text{endState}(w)$ ***is*** a Final State, otherwise it is rejected.

In a NFA:

- Any string w traces a ***set of paths***, starting from the Start State and ending at some ***set of states***, which we'll call $\text{endStates}(w)$.
 - The set might have zero, one or more members.
- The string w is accepted if $\text{endStates}(w)$ ***contains*** a Final State, otherwise it is rejected.



$\text{endStates(ab)} = \{1,3\}$

$\text{endStates(aba)} = \{1,2\}$

In general, if w is a string and x is a single letter, then
 $\text{endStates}(wx) =$

{ $q : \text{for some state } p \text{ in } \text{endStates}(w),$
 there is a transition $p \xrightarrow{x} q$ }

... provided there are no empty string transitions $q_1 \xrightarrow{\epsilon} q_2$

Converting a NFA to a FA

This suggests an algorithm for constructing all possible $\text{endStates}(w)$ for all strings w .

$w := \epsilon$

$\text{endStates}(\epsilon) := \{ \text{Start State} \}$

For all strings w in order of increasing length:

For each x in $\{a,b\}$

For each p in $\text{endStates}(w)$, and each transition $p \xrightarrow{x} q$

Add q to the set $\text{endStates}(wx)$.

... until we keep getting the same $\text{endStates}(\dots)$ sets all the time.

Again, assumes **there are no empty string transitions** $q_1 \xrightarrow{\epsilon} q_2$

Converting a NFA to a FA

Use the sets `endStates(...)` as the states of a new FA.

Transitions: $\text{endStates}(w) \xrightarrow{x} \text{endStates}(wx)$

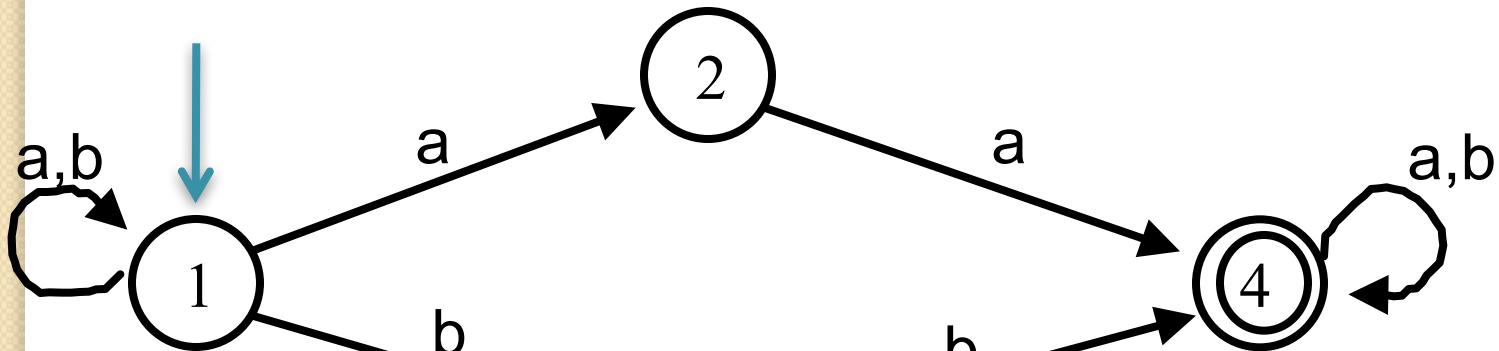
Start State of the new FA = { Start State of the NFA }

Final States of the new FA = any set `endStates(...)`
that contains a Final State of the NFA.

Algorithm is only an outline.

Some things to think about:

- loop through the p outside, then loop through x ?
- How do we know it stops?
- Complexity?
- How to deal with empty string transitions?



	a	b
Start {1}	{1,2}	{1,3}
{1,2}	{1,2,4}	{1,3}
{1,3}	{1,2}	{1,3,4}
Final {1,2,4}	{1,2,4}	{1,3,4}
Final {1,3,4}	{1,2,4}	{1,3,4}

Converting a NFA to a FA

Now suppose that the FA might have empty string transitions, $q_1 \xrightarrow{\epsilon} q_2$.

These allow change of state without reading any letter of the input string.

Every time we include a new state q in some `endStates(...)`, we also need to include any state we can reach *from it* along empty string transitions.

Look at all paths from q that just use ϵ transitions ...

$$q \xrightarrow{\epsilon} q_1 \xrightarrow{\epsilon} q_2 \xrightarrow{\epsilon} \dots \xrightarrow{\epsilon} q_i$$

... and include all states on such paths.

Modify earlier algorithm, for constructing the sets `endStates(...)`, to take account of empty string transitions.

Earlier algorithm:

$w := \epsilon$

`endStates(ϵ) := { Start State }`

For all strings w in order of increasing length:

For each x in $\{a,b\}$

For each p in `endStates(w)`, and each transition $p \xrightarrow{x} q$

Add q to the set `endStates(wx)`.

... until we keep getting the same `endStates(...)` sets all the time.

Modify earlier algorithm, for constructing the sets $\text{endStates}(\dots)$, to take account of empty string transitions.

Modified algorithm:

$w := \epsilon$

$\text{endStates}(\epsilon) := \{ \text{Start State} \}$

Add, to $\text{endStates}(\epsilon)$, states reachable from **Start State** along ϵ -transitions.

For all strings w in order of increasing length:

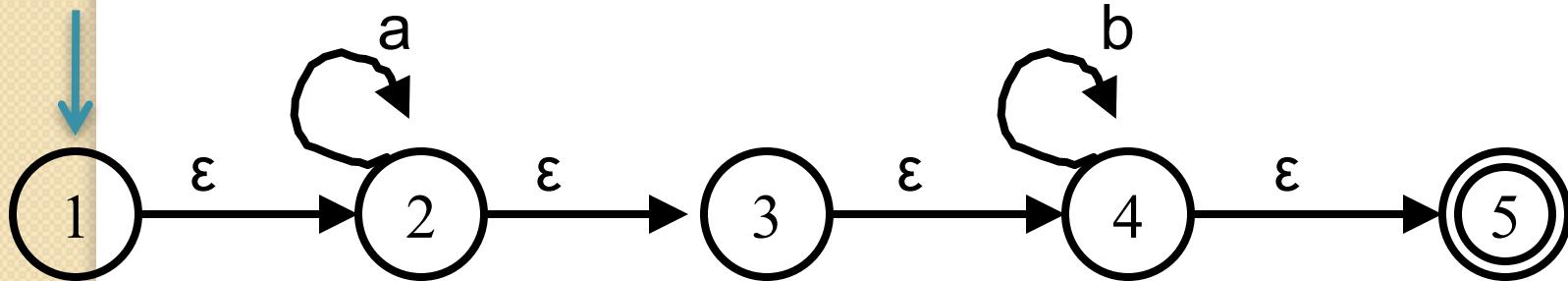
For each x in $\{a,b\}$

For each p in $\text{endStates}(w)$, and each transition $p \xrightarrow{x} q$

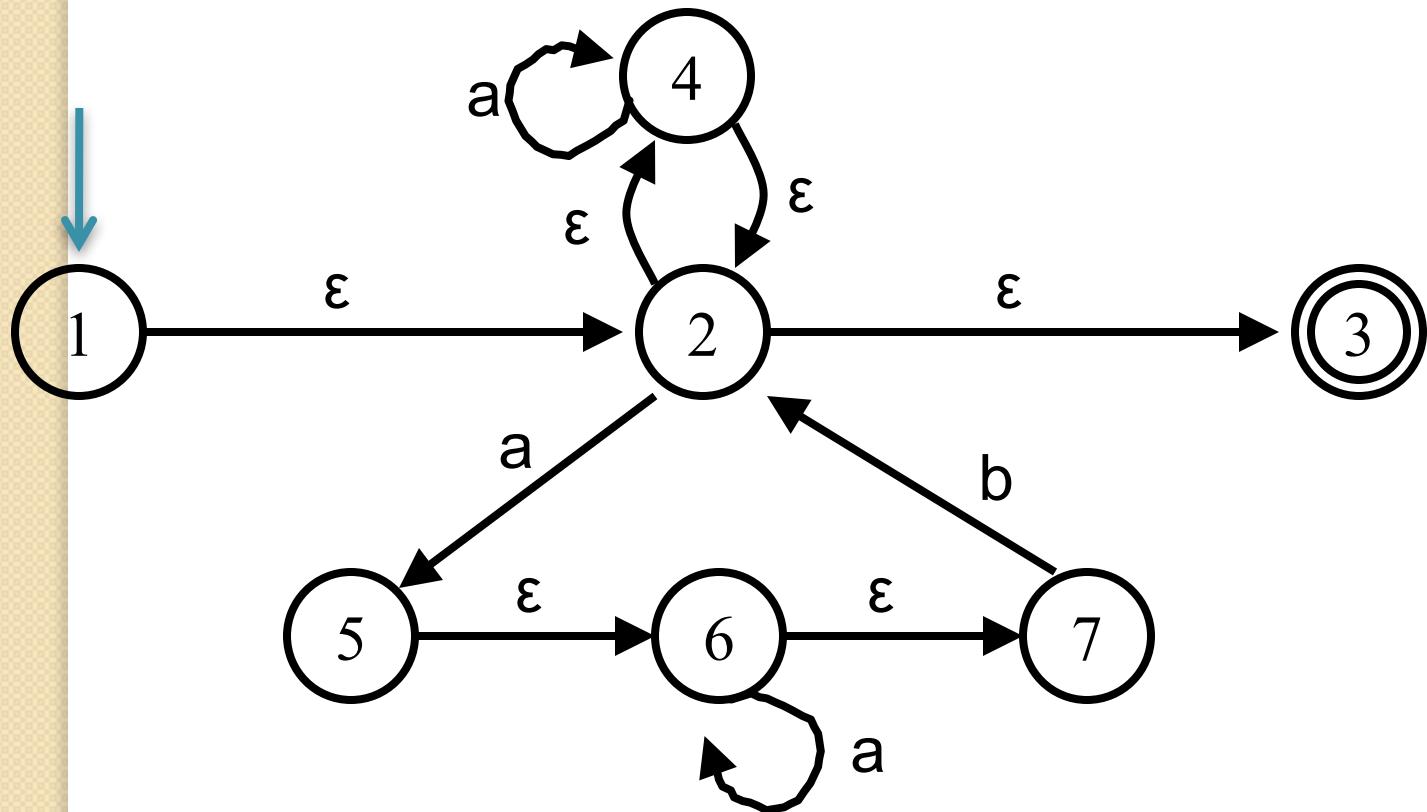
Add q to the set $\text{endStates}(wx)$.

Add, to $\text{endStates}(wx)$, states reachable from q along ϵ -transitions.

... until we keep getting the same $\text{endStates}(\dots)$ sets all the time.



	a	b
Start/Final	$\{1,2,3,4,5\}$	$\{2,3,4,5\}$
Final	$\{2,3,4,5\}$	$\{4,5\}$
Final	$\{4,5\}$	\emptyset
	\emptyset	\emptyset



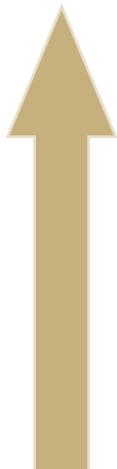
	a	b
Start /Final {1,2,3,4}	{5,4,6,7,2,3}	ϕ
Final {2,3,4,5,6,7}	{5,4,6,7,2,3}	{2,3,4}
Final {2,3,4}	{5,4,6,7,2,3}	ϕ
	ϕ	ϕ

Kleene's Theorem

Regular Expression



NFA



GNFA



Finite Automaton

Generalised Nondeterministic Finite Automaton (GNFA)

Definition

A **Generalised Nondeterministic Finite Automaton** (GNFA) is a NFA in which:

- transitions are labelled by *regular expressions*, not just by single letters;
- there is just one Final State, and it is not the Start State;
- there are transitions from every state to every other state (including itself), except that the Start State has no incoming transitions and the Final State has no outgoing transitions.

(Note: you can just label a transition by \emptyset if you don't want to use it.)

Generalised Nondeterministic Finite Automaton (GNFA)

Definition

A string w is **accepted** by a given GNFA if it can be divided into substrings, $w = w_1 \dots w_k$, 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, \dots, 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**.

From FA to GNFA

Given a FA:

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.

3. Add new arcs “everywhere”, labelled \emptyset .

The letters on the arcs are already regular expressions in their own right.

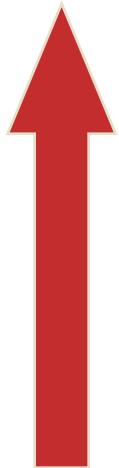
Now it's a GNFA, accepting the same language as the original FA.

Kleene's Theorem

Regular Expression



NFA



GNFA

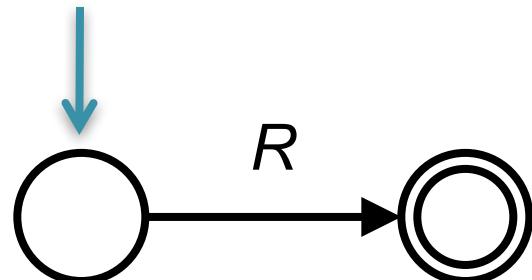


Finite Automaton

From GNFA to Regular Expression

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

We keep doing this until we have a GNFA with just one transition:



If our initial GNFA has only a Start State and a Final State, we are done.

From GNFA to Regular Expression

So, let q be some non-Start, non-Final state.

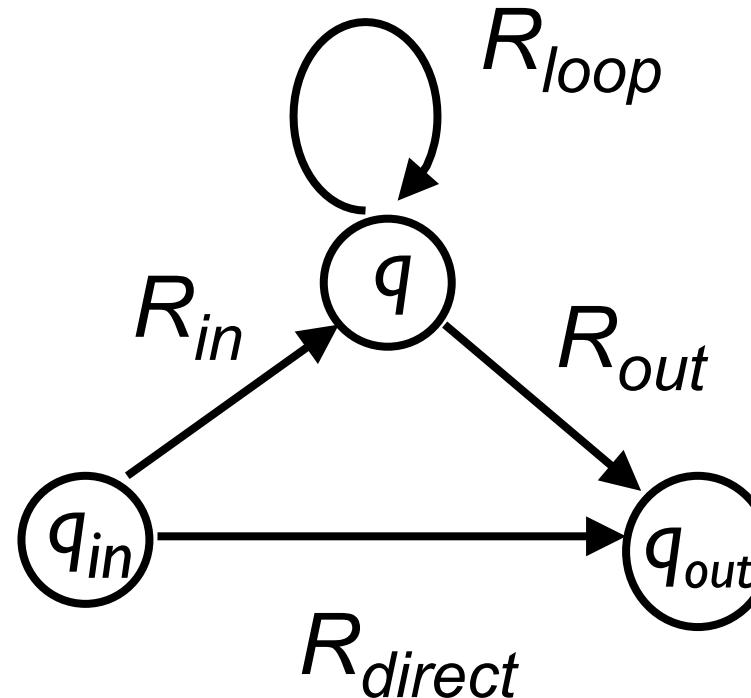
Let q_{in} be any non-Final state. Let R_{in} be the regular expression on the transition from q_{in} to q .

Let q_{out} be any non-Start state. Let R_{out} be the regular expression on the transition from q to q_{out} .

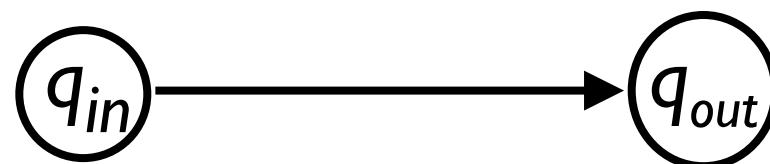
Let R_{loop} be the regular expression on the transition from q to itself.

Let R_{direct} be the regular expression on the transition from q_{in} to q_{out} .

From GNFA to Regular Expression



becomes ...



$$R_{direct} \cup (R_{in} (R_{loop})^* R_{out})$$

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

For FIT2004 students:

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

Revision

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

Reference

- Sipser, Ch I, especially pp 54-56, 66-76.