

# Compilers and Formal Languages (3)

Email: christian.urban at kcl.ac.uk

Office: S1.27 (1st floor Strand Building)

Slides: KEATS (also home work and course-work is there)

# Scala Book, Exams

- [www.inf.kcl.ac.uk/urbanc/ProgInScala2ed.pdf](http://www.inf.kcl.ac.uk/urbanc/ProgInScala2ed.pdf)
- homeworks (exam 80%)
- coursework (20%)

# Regular Expressions

In programming languages they are often used to recognise:

- symbols, digits
- identifiers
- numbers (non-leading zeros)
- keywords
- comments

<http://www.regexper.com>

# Last Week

Last week I showed you a regular expression matcher that works provably correct in all cases (we only started with the proving part though)

*matches*  $s r$  if and only if  $s \in L(r)$

by Janusz Brzozowski (1964)

# The Derivative of a Rexp

$$\mathit{der} c (\mathbf{0}) \stackrel{\text{def}}{=} \mathbf{0}$$

$$\mathit{der} c (\mathbf{1}) \stackrel{\text{def}}{=} \mathbf{0}$$

$$\mathit{der} c (d) \stackrel{\text{def}}{=} \text{if } c = d \text{ then } \mathbf{1} \text{ else } \mathbf{0}$$

$$\mathit{der} c (r_1 + r_2) \stackrel{\text{def}}{=} \mathit{der} c r_1 + \mathit{der} c r_2$$

$$\mathit{der} c (r_1 \cdot r_2) \stackrel{\text{def}}{=} \text{if } \mathit{nullable}(r_1) \\ \text{then } (\mathit{der} c r_1) \cdot r_2 + \mathit{der} c r_2 \\ \text{else } (\mathit{der} c r_1) \cdot r_2$$

$$\mathit{der} c (r^*) \stackrel{\text{def}}{=} (\mathit{der} c r) \cdot (r^*)$$

$$\mathit{ders} [] r \stackrel{\text{def}}{=} r$$

$$\mathit{ders} (c :: s) r \stackrel{\text{def}}{=} \mathit{ders} s (\mathit{der} c r)$$

Input: string *abc* and regular expression *r*

- 1 *der a r*
- 2 *der b (der a r)*
- 3 *der c (der b (der a r))*

Input: string *abc* and regular expression *r*

- 1 *der a r*
- 2 *der b (der a r)*
- 3 *der c (der b (der a r))*
- 4 finally check whether the last regular expression can match the empty string

We proved already

*nullable*( $r$ ) if and only if  $\epsilon \in L(r)$

by induction on the regular expression  $r$ .



We proved already

*nullable*( $r$ ) if and only if  $\epsilon \in L(r)$

by induction on the regular expression  $r$ .

**Any Questions?**

We need to prove

$$L(\text{der } c r) = \text{Der } c (L(r))$$

also by induction on the regular expression  $r$ .

# Proofs about Rexp

- $P$  holds for  $\mathbf{0}$ ,  $\mathbf{1}$  and  $c$
- $P$  holds for  $r_1 + r_2$  under the assumption that  $P$  already holds for  $r_1$  and  $r_2$ .
- $P$  holds for  $r_1 \cdot r_2$  under the assumption that  $P$  already holds for  $r_1$  and  $r_2$ .
- $P$  holds for  $r^*$  under the assumption that  $P$  already holds for  $r$ .

# Proofs about Natural Numbers and Strings

- $P$  holds for  $0$  and
- $P$  holds for  $n + 1$  under the assumption that  $P$  already holds for  $n$
  
- $P$  holds for  $[]$  and
- $P$  holds for  $c::s$  under the assumption that  $P$  already holds for  $s$

# Regular Expressions

$r ::= \mathbf{0}$	null
$\mathbf{1}$	empty string / "" / []
$c$	character
$r_1 \cdot r_2$	sequence
$r_1 + r_2$	alternative / choice
$r^*$	star (zero or more)

How about ranges  $[a-z]$ ,  $r^+$  and  $\sim r$ ? Do they increase the set of languages we can recognise?

# Negation of Regular Expr's

- $\sim r$  (everything that  $r$  cannot recognise)
- $L(\sim r) \stackrel{\text{def}}{=} UNIV - L(r)$
- $nullable(\sim r) \stackrel{\text{def}}{=} \text{not } (nullable(r))$
- $derc(\sim r) \stackrel{\text{def}}{=} \sim (derc r)$

# Negation of Regular Expr's

- $\sim r$  (everything that  $r$  cannot recognise)
- $L(\sim r) \stackrel{\text{def}}{=} UNIV - L(r)$
- $nullable(\sim r) \stackrel{\text{def}}{=} \text{not } (nullable(r))$
- $derc(\sim r) \stackrel{\text{def}}{=} \sim (derc r)$

Used often for recognising comments:

$$/ \cdot * \cdot (\sim ([a-z]^* \cdot * \cdot / \cdot [a-z]^*)) \cdot * \cdot /$$

# Negation

Assume you have an alphabet consisting of the letters *a*, *b* and *c* only. Find a (basic!) regular expression that matches all strings *except* *ab* and *ac*!



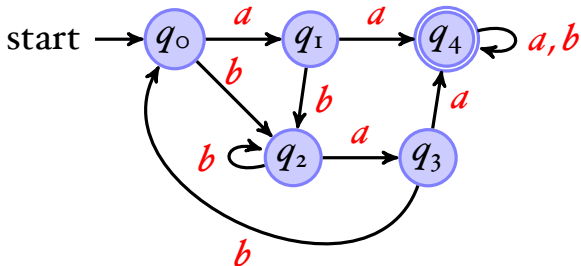
# Automata

A **deterministic finite automaton**, DFA, consists of:

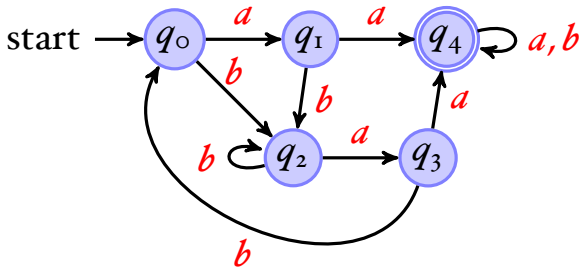
- a set of states  $\mathcal{Q}$
- one of these states is the start state  $q_0$
- some states are accepting states  $F$ , and
- there is transition function  $\delta$

which takes a state as argument and a character and produces a new state; this function might not be everywhere defined

$$A(\mathcal{Q}, q_0, F, \delta)$$



- the start state can be an accepting state
- it is possible that there is no accepting state
- all states might be accepting (but this does not necessarily mean all strings are accepted)



for this automaton  $\delta$  is the function

$$\begin{array}{lll}
 (q_0, a) \rightarrow q_1 & (q_1, a) \rightarrow q_4 & (q_4, a) \rightarrow q_4 \\
 (q_0, b) \rightarrow q_2 & (q_1, b) \rightarrow q_2 & (q_4, b) \rightarrow q_4 \dots
 \end{array}$$

# Accepting a String

Given

$$A(\mathcal{Q}, q_0, F, \delta)$$

you can define

$$\begin{aligned}\hat{\delta}(q, []) &\stackrel{\text{def}}{=} q \\ \hat{\delta}(q, c :: s) &\stackrel{\text{def}}{=} \hat{\delta}(\delta(q, c), s)\end{aligned}$$

# Accepting a String

Given

$$A(\mathcal{Q}, q_0, F, \delta)$$

you can define

$$\begin{aligned}\hat{\delta}(q, []) &\stackrel{\text{def}}{=} q \\ \hat{\delta}(q, c :: s) &\stackrel{\text{def}}{=} \hat{\delta}(\delta(q, c), s)\end{aligned}$$

Whether a string  $s$  is accepted by  $A$ ?

$$\hat{\delta}(q_0, s) \in F$$

# Regular Languages

A **language** is a set of strings.

A **regular expression** specifies a language.

A language is **regular** iff there exists a regular expression that recognises all its strings.

# Regular Languages

A **language** is a set of strings.

A **regular expression** specifies a language.

A language is **regular** iff there exists a regular expression that recognises all its strings.

not all languages are regular, e.g.  $a^n b^n$  is not

# Regular Languages (2)

A language is **regular** iff there exists a regular expression that recognises all its strings.

or **equivalently**

A language is **regular** iff there exists a deterministic finite automaton that recognises all its strings.



# Non-Deterministic Finite Automata

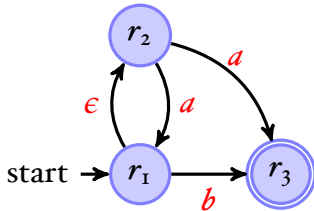
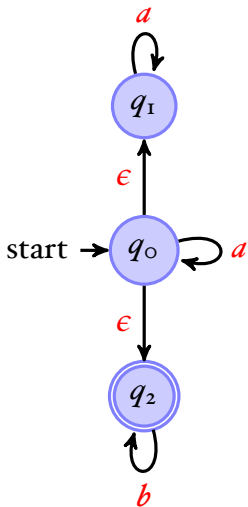
A non-deterministic finite automaton consists again of:

- a finite set of states
- one of these states is the start state
- some states are accepting states, and
- there is transition **relation**

$$\begin{aligned}(q_1, a) &\rightarrow q_2 \\ (q_1, a) &\rightarrow q_3\end{aligned}$$

$$(q_1, \epsilon) \rightarrow q_2$$

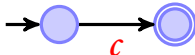
# Two NFA Examples



# Rexp to NFA

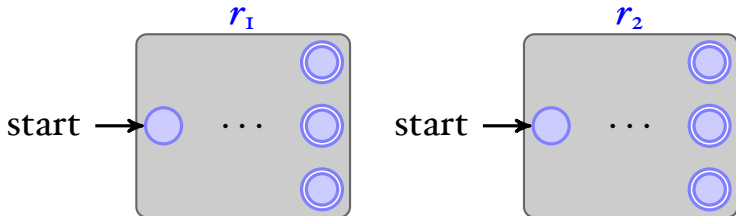
**o** start → 

**1** start → 

**c** start → 

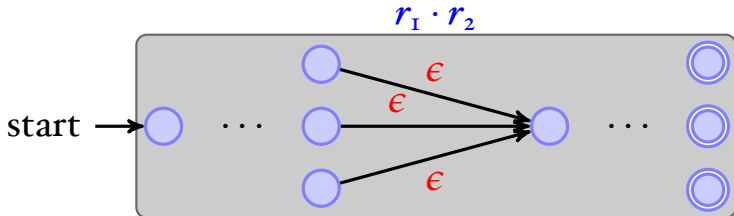
# Case $r_1 \cdot r_2$

By recursion we are given two automata:



We need to (1) change the accepting nodes of the first automaton into non-accepting nodes, and (2) connect them via  $\epsilon$ -transitions to the starting state of the second automaton.

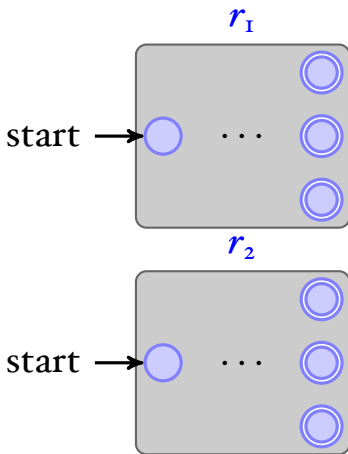
# Case $r_1 \cdot r_2$



We need to (1) change the accepting nodes of the first automaton into non-accepting nodes, and (2) connect them via  $\epsilon$ -transitions to the starting state of the second automaton.

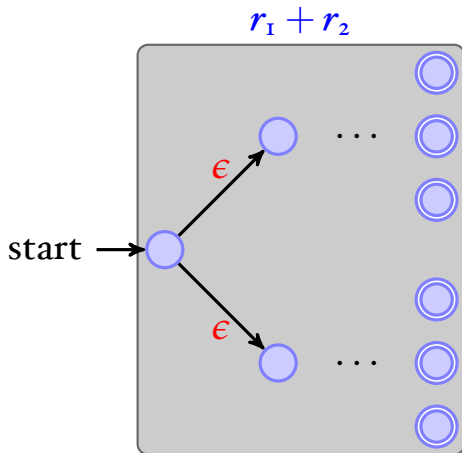
# Case $r_1 + r_2$

By recursion we are given two automata:



We (1) need to introduce a new starting state and (2) connect it to the original two starting states.

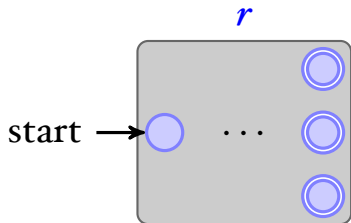
# Case $r_1 + r_2$



We (1) need to introduce a new starting state and (2) connect it to the original two starting states.

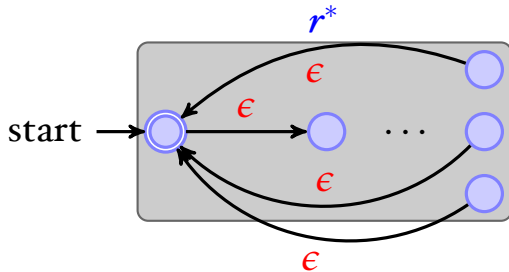
# Case $r^*$

By recursion we are given an automaton for  $r$ :

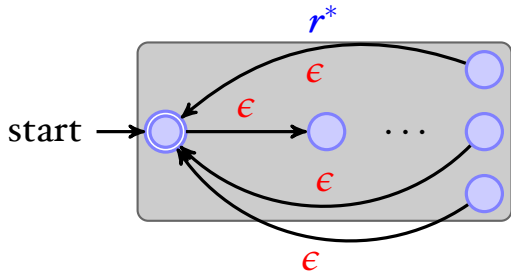




# Case $r^*$

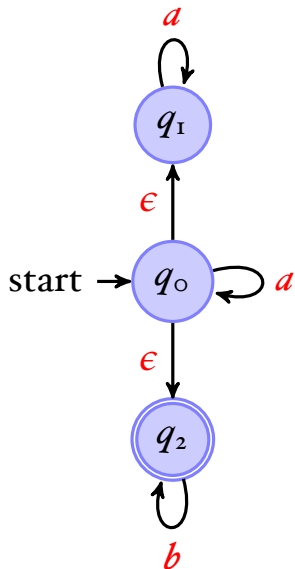


# Case $r^*$



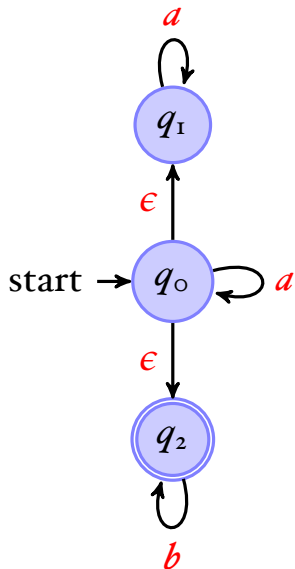
Why can't we just have an epsilon transition from the accepting states to the starting state?

# Subset Construction



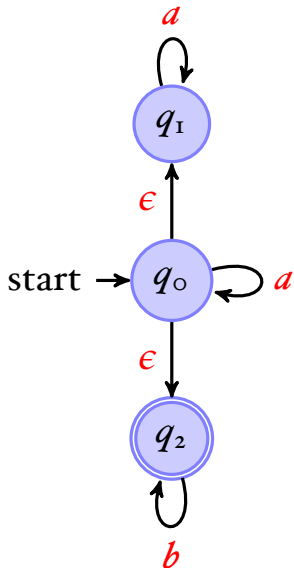
nodes	$a$	$b$
$\{\}$		
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

# Subset Construction



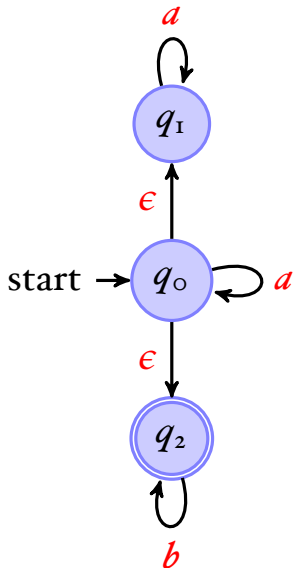
nodes	$a$	$b$
$\{\}$	$\{\}$	$\{\}$
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

# Subset Construction



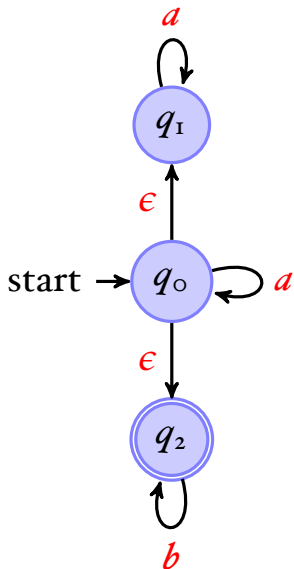
nodes	$a$	$b$
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

# Subset Construction



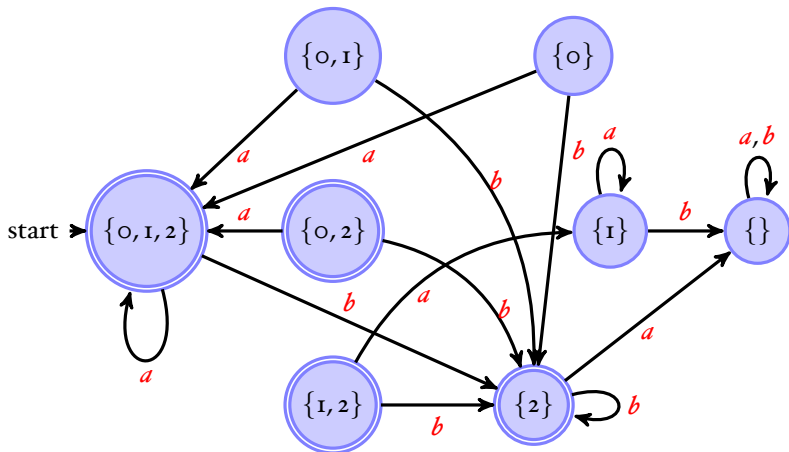
nodes	$a$	$b$
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0, 1\}$	$\{0, 1, 2\}$	$\{2\}$
$\{0, 2\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1, 2\}$	$\{1\}$	$\{2\}$
$\{0, 1, 2\}$	$\{0, 1, 2\}$	$\{2\}$

# Subset Construction



nodes	$a$	$b$
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$ *	$\{\}$	$\{2\}$
$\{0, 1\}$	$\{0, 1, 2\}$	$\{2\}$
$\{0, 2\}$ *	$\{0, 1, 2\}$	$\{2\}$
$\{1, 2\}$ *	$\{1\}$	$\{2\}$
s: $\{0, 1, 2\}$ *	$\{0, 1, 2\}$	$\{2\}$

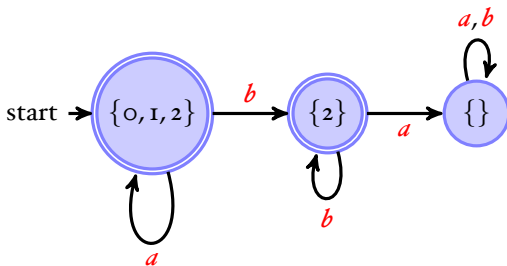
# The Result



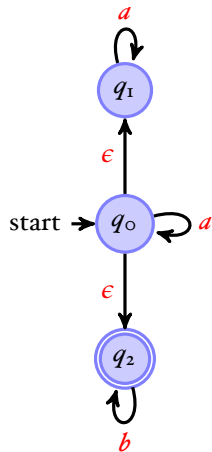


# Removing Dead States

DFA:



(original) NFA:



# Regexps and Automata

Thompson's construction      subset construction

**Regexps**  **NFAs**  **DFAs**

# Regexps and Automata

Thompson's construction      subset construction

**Regexps** → **NFAs** → **DFAs** → **minimal DFAs**

minimisation

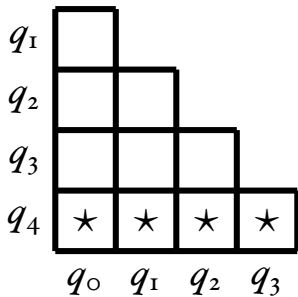
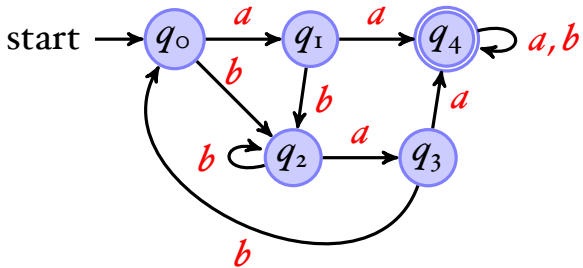
# DFA Minimisation

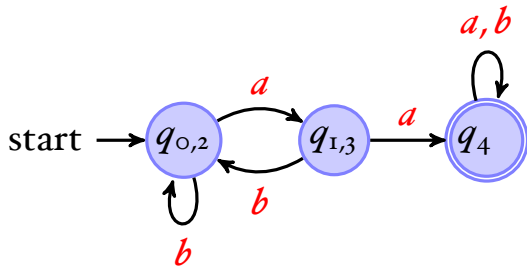
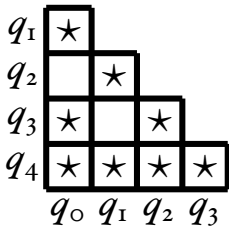
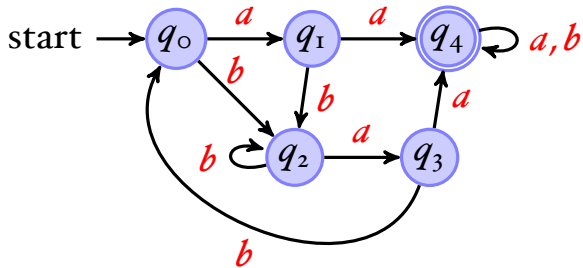
- 1 Take all pairs  $(q, p)$  with  $q \neq p$
- 2 Mark all pairs that accepting and non-accepting states
- 3 For all unmarked pairs  $(q, p)$  and all characters  $c$  test whether

$$(\delta(q, c), \delta(p, c))$$

are marked. If yes in at least one case, then also mark  $(q, p)$ .

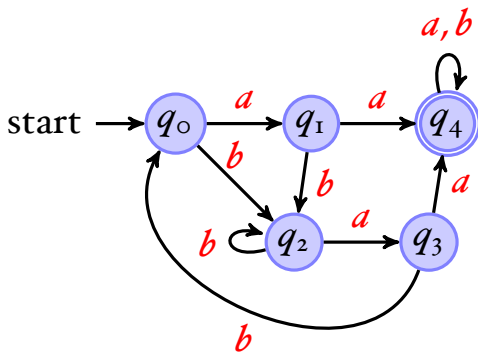
- 4 Repeat last step until no change.
- 5 All unmarked pairs can be merged.



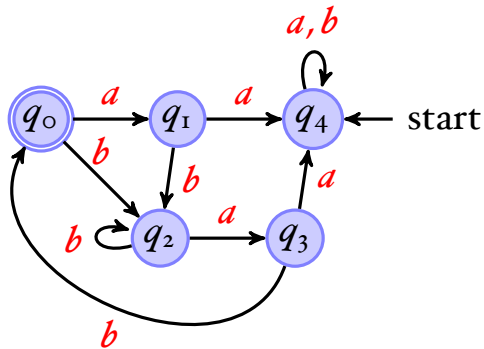


minimal automaton

# Alternatives



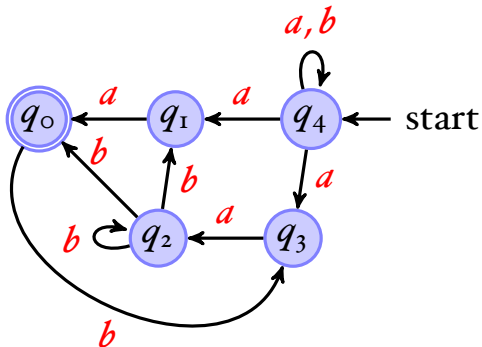
# Alternatives



- exchange initial / accepting states

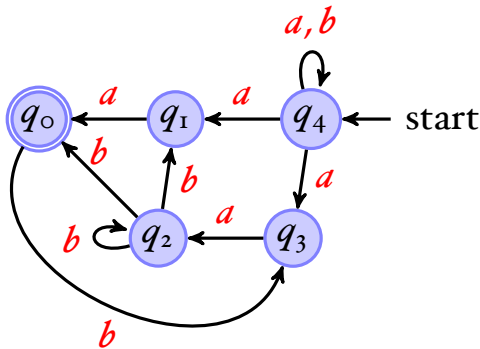


# Alternatives



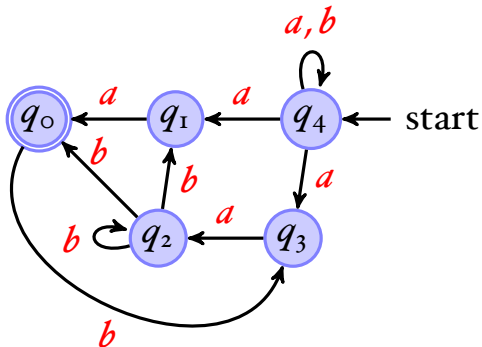
- exchange initial / accepting states
- reverse all edges

# Alternatives



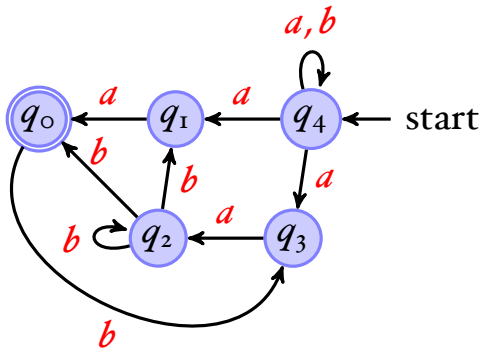
- exchange initial / accepting states
- reverse all edges
- subset construction  $\Rightarrow$  DFA

# Alternatives



- exchange initial / accepting states
- reverse all edges
- subset construction  $\Rightarrow$  DFA
- remove dead states

# Alternatives



- exchange initial / accepting states
- reverse all edges
- subset construction  $\Rightarrow$  DFA
- remove dead states
- repeat once more  $\Rightarrow$  minimal DFA

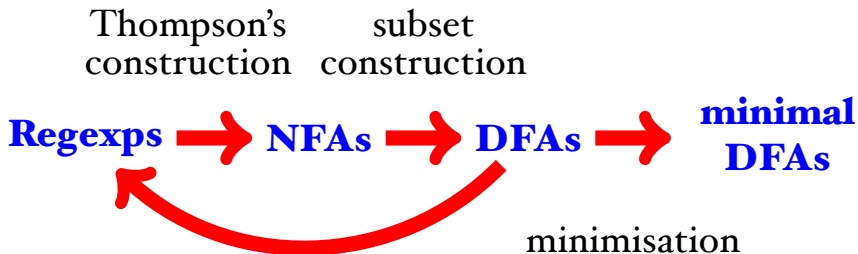
# Regexps and Automata

Thompson's construction      subset construction

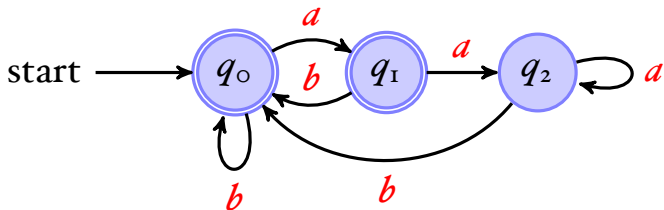
**Regexps**  **NFAs**  **DFAs**  **minimal DFAs**

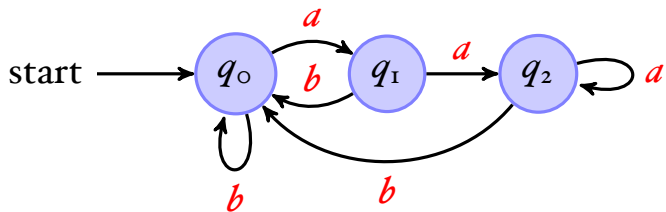
minimisation

# Regexps and Automata

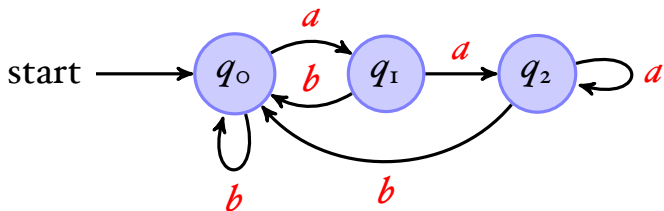


# DFA to Rexp







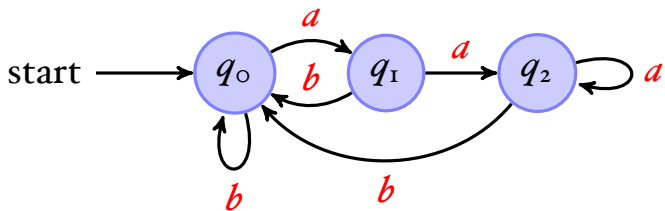


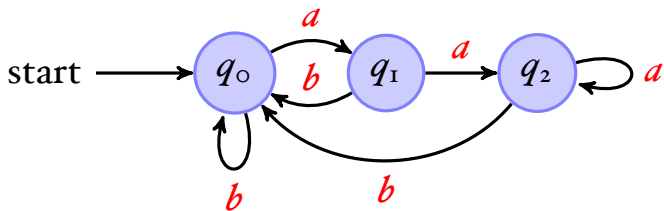
You know how to solve since school days, no?

$$q_0 = 2q_0 + 3q_1 + 4q_2$$

$$q_1 = 2q_0 + 3q_1 + 1q_2$$

$$q_2 = 1q_0 + 5q_1 + 2q_2$$

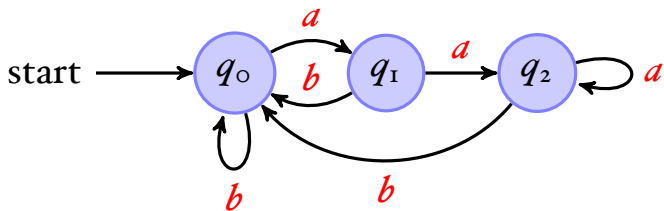




$$q_0 = \mathbf{I} + q_0 b + q_1 b + q_2 b$$

$$q_1 = q_0 a$$

$$q_2 = q_1 a + q_2 a$$



$$q_0 = \mathbf{I} + q_0 b + q_1 b + q_2 b$$

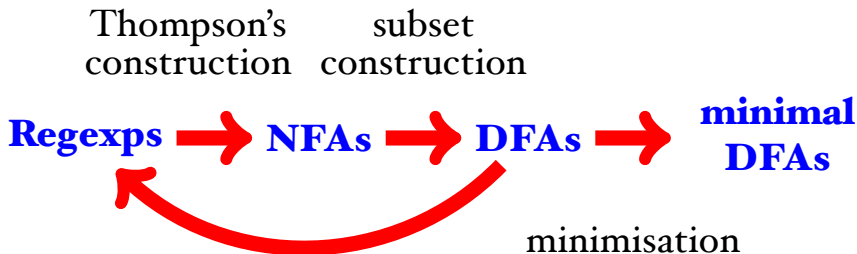
$$q_1 = q_0 a$$

$$q_2 = q_1 a + q_2 a$$

Arden's Lemma:

$$\text{If } q = qr + s \text{ then } q = sr^*$$

# Regexps and Automata



# Regular Languages (3)

A language is **regular** iff there exists a regular expression that recognises all its strings.

or **equivalently**

A language is **regular** iff there exists a deterministic finite automaton that recognises all its strings.

# Regular Languages (3)

A language is **regular** iff there exists a regular expression that recognises all its strings.

or **equivalently**

A language is **regular** iff there exists a deterministic finite automaton that recognises all its strings.

Why is every finite set of strings a regular language?

Given the function

$$\mathit{rev}(\mathbf{0}) \stackrel{\text{def}}{=} \mathbf{0}$$

$$\mathit{rev}(\mathbf{I}) \stackrel{\text{def}}{=} \mathbf{I}$$

$$\mathit{rev}(c) \stackrel{\text{def}}{=} c$$

$$\mathit{rev}(r_1 + r_2) \stackrel{\text{def}}{=} \mathit{rev}(r_1) + \mathit{rev}(r_2)$$

$$\mathit{rev}(r_1 \cdot r_2) \stackrel{\text{def}}{=} \mathit{rev}(r_2) \cdot \mathit{rev}(r_1)$$

$$\mathit{rev}(r^*) \stackrel{\text{def}}{=} \mathit{rev}(r)^*$$

and the set

$$\mathit{Rev} A \stackrel{\text{def}}{=} \{s^{-1} \mid s \in A\}$$

prove whether

$$L(\mathit{rev}(r)) = \mathit{Rev}(L(r))$$