Automata 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)

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)

$$\mathit{matches}\, s\, r \quad \text{if and only if} \quad s \in L(r)$$

by Janusz Brzozowski (1964)

The Derivative of a Rexp

$$
\begin{array}{lcl}
der\,c\,(\varnothing) & \stackrel{\text{def}}{=} & \varnothing \\
der\,c\,(\epsilon) & \stackrel{\text{def}}{=} & \varnothing \\
der\,c\,(d) & \stackrel{\text{def}}{=} & \text{if } c = d \text{ then } \epsilon \text{ else } \varnothing \\
der\,c\,(r_1 + r_2) & \stackrel{\text{def}}{=} & der\,c\,r_1 + der\,c\,r_2 \\
der\,c\,(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \text{if } nullable(r_1) \\
 & & \text{then } (der\,c\,r_1) \cdot r_2 + der\,c\,r_2 \\
 & & \text{else } (der\,c\,r_1) \cdot r_2 \\
der\,c\,(r^*) & \stackrel{\text{def}}{=} & (der\,c\,r) \cdot (r^*) \\
 & & \\
ders\,[]\,r & \stackrel{\text{def}}{=} & r \\
ders\,(c::s)\,r & \stackrel{\text{def}}{=} & ders\,s\,(der\,c\,r)
\end{array}
$$

To see what is going on, define

$$Der\,c\,A \stackrel{\text{def}}{=} \{s \mid c::s \in A\}$$

For $A = \{foo, bar, frak\}$ then

$$\begin{array}{lcl} Der\,f\,A & = & \{oo, rak\} \\ Der\,b\,A & = & \{ar\} \\ Der\,a\,A & = & \varnothing \end{array}$$

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r then

1. $Der\,a\,(L(r))$

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r then

1. $Der\, a\, (L(r))$
2. $Der\, b\, (Der\, a\, (L(r)))$

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r then

1. $Der\, a\, (L(r))$
2. $Der\, b\, (Der\, a\, (L(r)))$
3. $Der\, c\, (Der\, b\, (Der\, a\, (L(r))))$

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r then

1. $Der\, a\, (L(r))$
2. $Der\, b\, (Der\, a\, (L(r)))$
3. $Der\, c\, (Der\, b\, (Der\, a\, (L(r))))$

4. finally we test whether the empty string is in this set

The Idea of the Algorithm

If we want to recognise the string abc with regular expression r then

1. $Der\,a\,(L(r))$
2. $Der\,b\,(Der\,a\,(L(r)))$
3. $Der\,c\,(Der\,b\,(Der\,a\,(L(r))))$

4. finally we test whether the empty string is in this set

The matching algorithm works similarly, just over regular expressions instead of sets.

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

$$nullable(r) \text{ if and only if } [] \in L(r)$$

by induction on the regular expression.

We proved already

$$nullable(r) \text{ if and only if } [] \in L(r)$$

by induction on the regular expression.

Any Questions?

We need to prove

$$L(der\,c\,r) = Der\,c\,(L(r))$$

by induction on the regular expression.

Proofs about Rexps

- P holds for $\varnothing$, ϵ and $\mathtt{c}$
- 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	$::=$	$\varnothing$	null
	$\mid$	ϵ	empty string / "" / []
	$\mid$	c	character
	$\mid$	$r_1 \cdot r_2$	sequence
	$\mid$	$r_1 + r_2$	alternative / choice
	$\mid$	r^*	star (zero or more)

How about ranges $[a\text{-}z]$, 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)
- $L(\sim r) \stackrel{\text{def}}{=} UNIV - L(r)$
- $nullable(\sim r) \stackrel{\text{def}}{=} not\,(nullable(r))$
- $der\,c\,(\sim r) \stackrel{\text{def}}{=} \sim (der\,c\,r)$

Negation of Regular Expr's

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

Used often for recognising comments:

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

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!

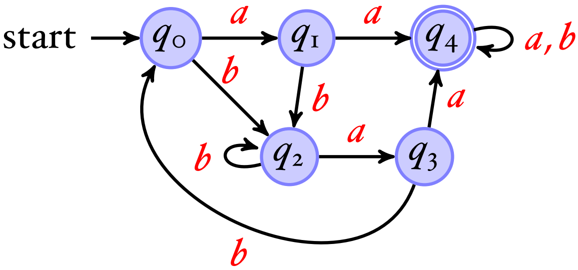
Automata

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

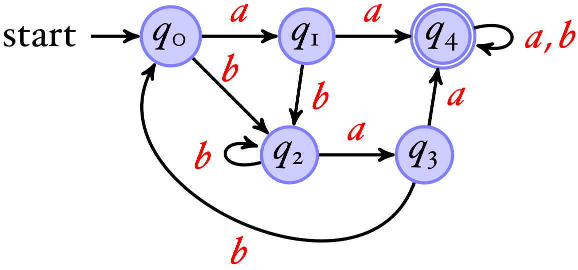
- a set of states $\mathcal{Q}$
- one of these states is the start state q_0
- some states are accepting states F, and
- there is transition function δ

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 δ is the function

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

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

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

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

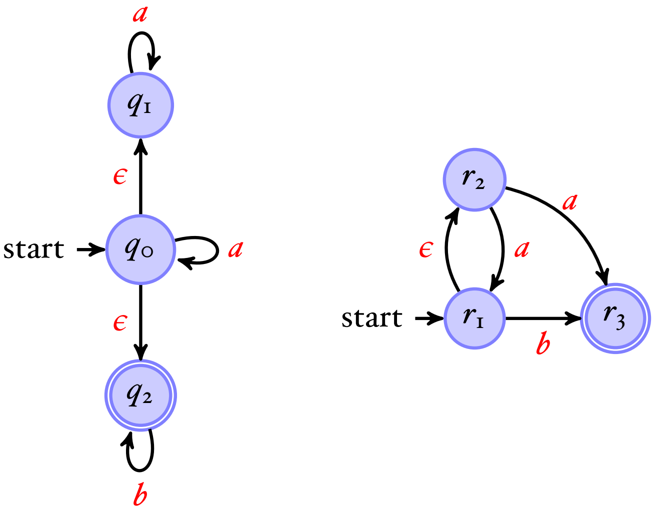
A non-deterministic finite automaton consists again of:

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

$$(q_1, a) \rightarrow q_2$$
$$(q_1, a) \rightarrow q_3$$

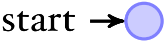
$$(q_1, \epsilon) \rightarrow q_2$$

Two NFA Examples

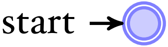


Rexp to NFA

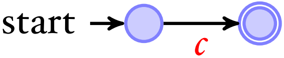
$\varnothing$



ϵ

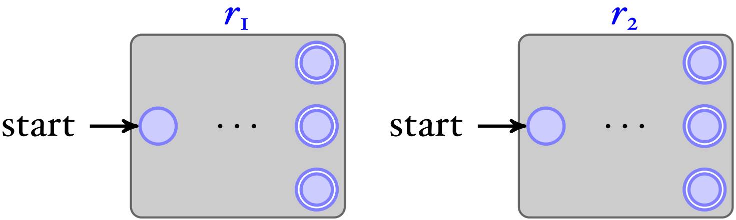


c



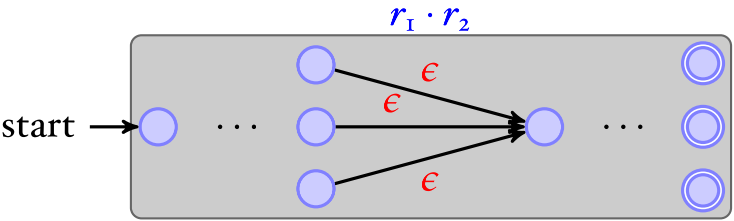
Case $r_1 \cdot r_2$

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 ϵ-transitions to the starting state of the second automaton.

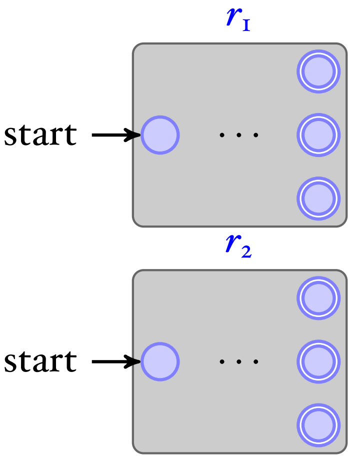
Case $r_1 \cdot r_2$



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

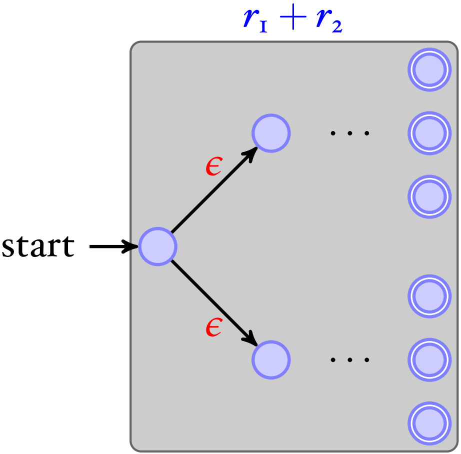
Case $r_1 + r_2$

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.

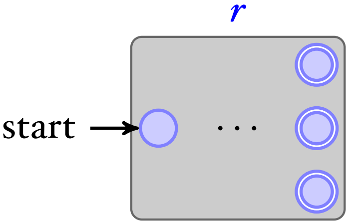
Case $r_1 + r_2$



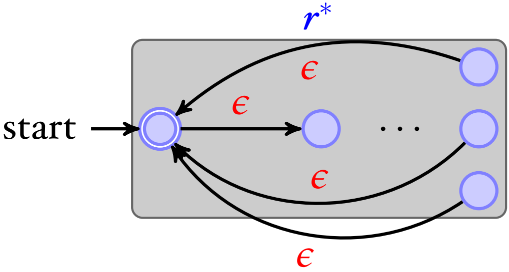
We (1) need to introduce a new starting state and (2) connect it to the original two starting states.

Case r^*

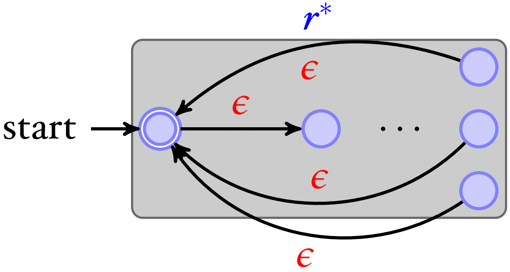
By recursion we are given an automaton for r:



Case r^*

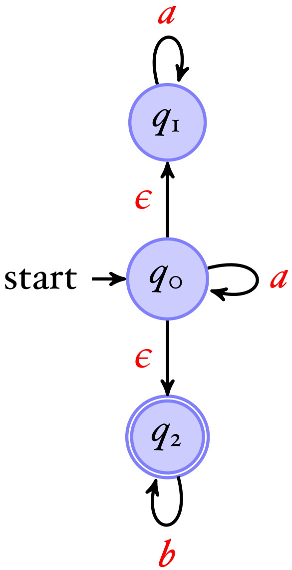


Case r^*



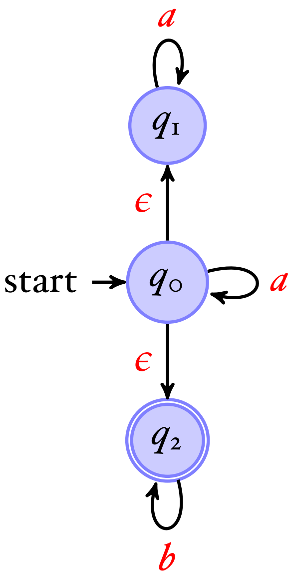
Why can't we just have an epsilon transition from the accepting states to the starting state?

Subset Construction



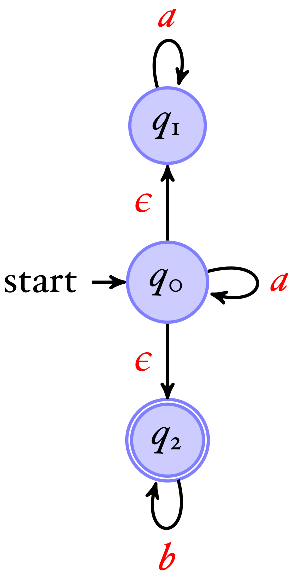
nodes	a	b
$\{\}$		
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



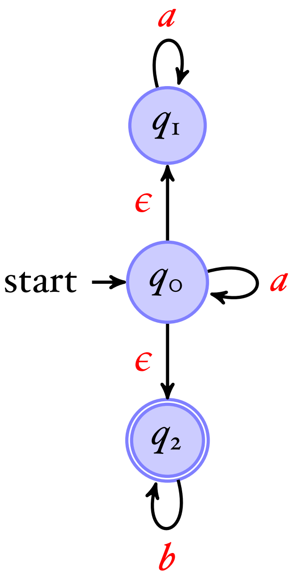
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



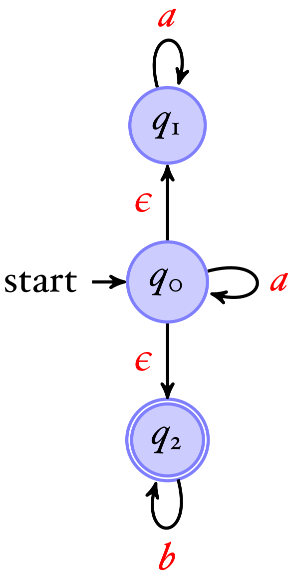
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

Subset Construction



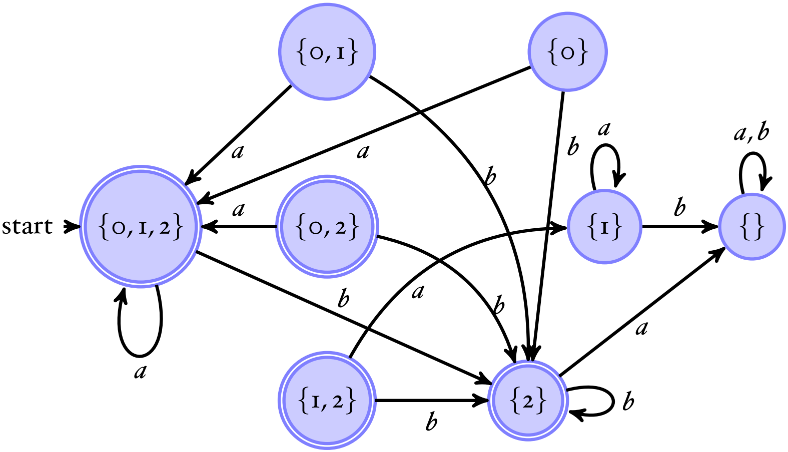
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0,1\}$	$\{0,1,2\}$	$\{2\}$
$\{0,2\}$	$\{0,1,2\}$	$\{2\}$
$\{1,2\}$	$\{1\}$	$\{2\}$
$\{0,1,2\}$	$\{0,1,2\}$	$\{2\}$

Subset Construction



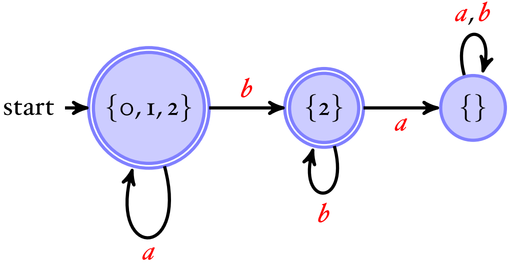
nodes		a	b
$\{\}$		$\{\}$	$\{\}$
$\{0\}$		$\{0,1,2\}$	$\{2\}$
$\{1\}$		$\{1\}$	$\{\}$
$\{2\}$	*	$\{\}$	$\{2\}$
$\{0,1\}$		$\{0,1,2\}$	$\{2\}$
$\{0,2\}$	*	$\{0,1,2\}$	$\{2\}$
$\{1,2\}$	*	$\{1\}$	$\{2\}$
s: $\{0,1,2\}$	*	$\{0,1,2\}$	$\{2\}$

The Result

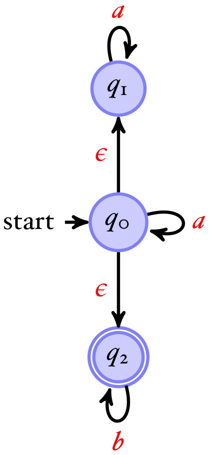


Removing Dead States

DFA:



NFA:



Regexps and Automata

Thompson's construction | subset construction

Regexps → **NFAs** → **DFAs**

Regexps and Automata

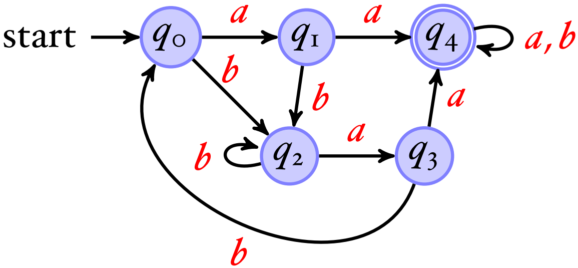
Thompson's construction | subset construction

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

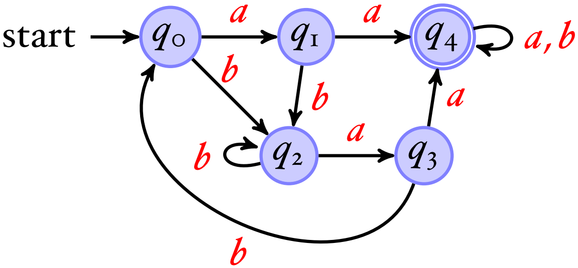
minimisation

DFA Minimisation

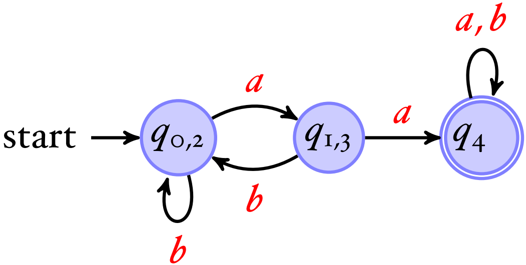
1. Take all pairs (q, p) with $q \neq p$
2. Mark all pairs that accepting and non-accepting states
3. For all unmarked pairs (q, p) and all characters c test whether
$$(\delta(q,c), \delta(p,c))$$
are marked. If yes, then also mark (q, p).
4. Repeat last step until no change.
5. All unmarked pairs can be merged.



q_1				
q_2				
q_3				
q_4	$\star$	$\star$	$\star$	$\star$
	q_0	q_1	q_2	q_3

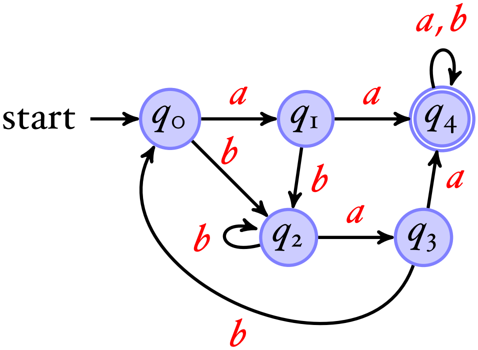


	q_0	q_1	q_2	q_3
q_1	★			
q_2		★		
q_3	★		★	
q_4	★	★	★	★

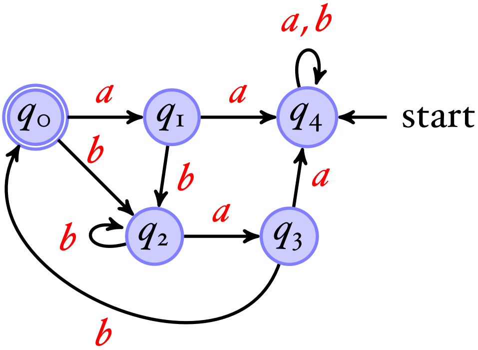


minimal automaton

Alternatives

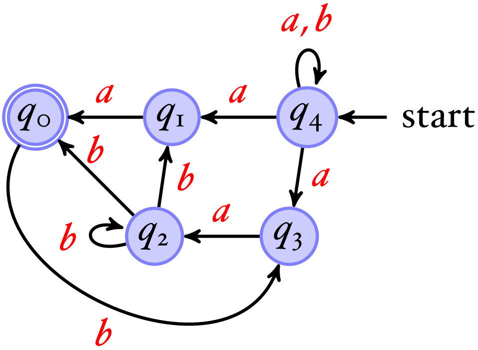


Alternatives



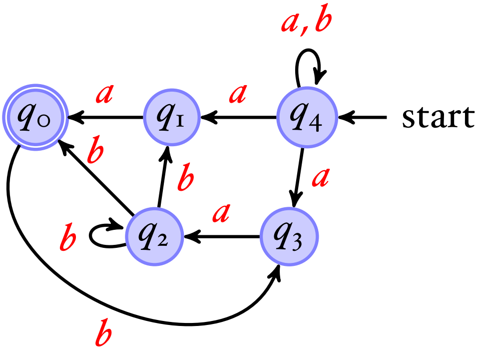
- exchange initial / accepting states

Alternatives



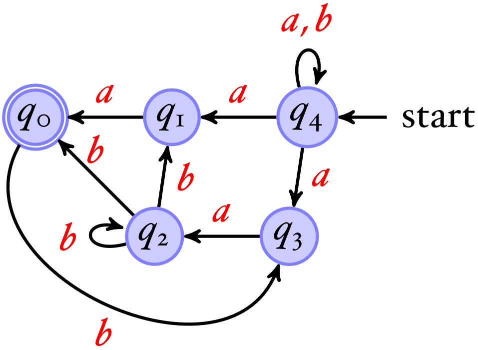
- exchange initial / accepting states
- reverse all edges

Alternatives



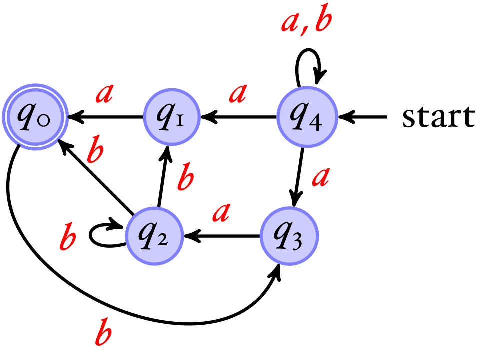
- exchange initial / accepting states
- reverse all edges
- subset construction ⇒ DFA

Alternatives



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

Alternatives



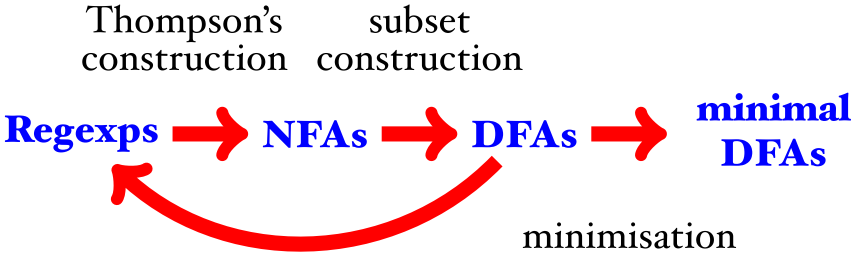
- exchange initial / accepting states
- reverse all edges
- subset construction ⇒ DFA
- remove dead states
- repeat once more ⇒ minimal DFA

Regexps and Automata

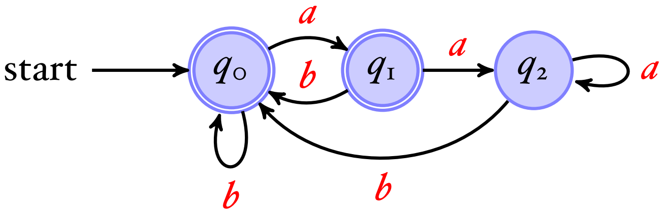
Thompson's construction — subset construction — minimisation

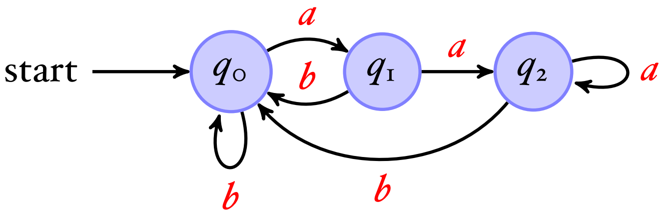
Regexps → **NFAs** → **DFAs** → **minimal DFAs**

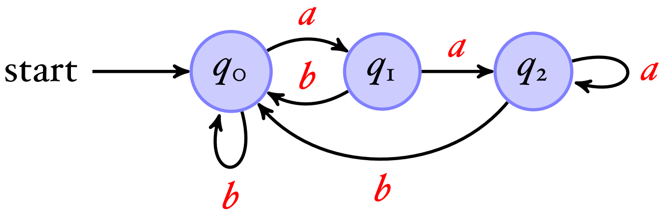
Regexps and Automata



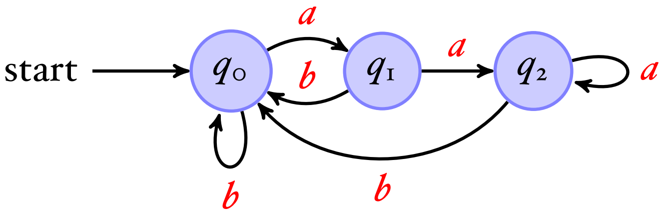
DFA to Rexp

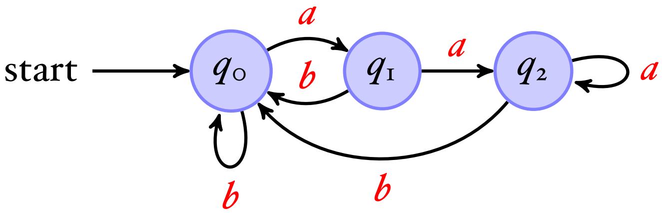



start
a
b
a
a
b
b
q_0
q_1
q_2

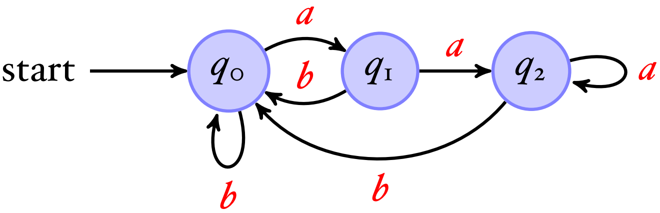


$$
\begin{aligned}
q_0 &= 2\,q_0 + 3\,q_1 + 4\,q_2 \\
q_1 &= 2\,q_0 + 3\,q_1 + 1\,q_2 \\
q_2 &= 1\,q_0 + 5\,q_1 + 2\,q_2
\end{aligned}
$$


start
q_0
q_1
q_2
a
b
a
a
b
b



$$
\begin{aligned}
q_0 &= \epsilon + q_0\,b + q_1\,b + q_2\,b \\
q_1 &= q_0\,a \\
q_2 &= q_1\,a + q_2\,a
\end{aligned}
$$

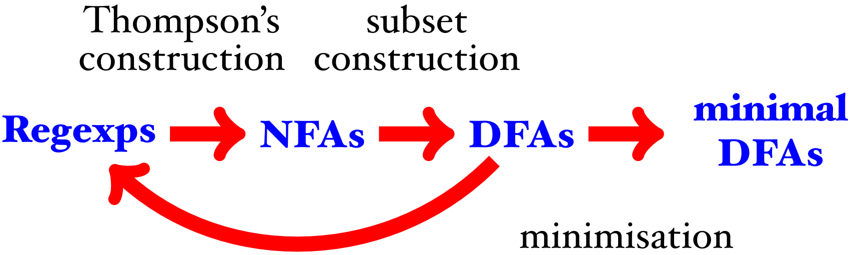


$$
\begin{aligned}
q_0 &= \epsilon + q_0\,b + q_1\,b + q_2\,b \\
q_1 &= q_0\,a \\
q_2 &= q_1\,a + q_2\,a
\end{aligned}
$$

Arden's Lemma:

If $q = q\,r + s$ then $q = s\,r^*$

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

$$\begin{aligned}
rev(\varnothing) &\stackrel{\text{def}}{=} \varnothing \\
rev(\epsilon) &\stackrel{\text{def}}{=} \epsilon \\
rev(c) &\stackrel{\text{def}}{=} c \\
rev(r_1 + r_2) &\stackrel{\text{def}}{=} rev(r_1) + rev(r_2) \\
rev(r_1 \cdot r_2) &\stackrel{\text{def}}{=} rev(r_2) \cdot rev(r_1) \\
rev(r^*) &\stackrel{\text{def}}{=} rev(r)^*
\end{aligned}$$

and the set

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

prove whether

$$L(rev(r)) = Rev(L(r))$$