Compilers and Formal Languages (3)

Email:	christian.urban at kcl.ac.uk
Office Hours:	Thursdays 12 – 14
Location:	N7.07 (North Wing, Bush House)
Slides & Progs:	KEATS (also homework is there)

Scala Book, Exams

- https://nms.kcl.ac.uk/christian.urban/ProgInScala2ed.pdf
- homework (written exam 80%)
- coursework (20%)

- short survey at KEATS; to be answered until Sunday

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)

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

by Janusz Brzozowski (1964)

The Derivative of a Rexp

$$
\begin{array}{lcl}
der\,c\,(\mathbf{0}) & \stackrel{\text{def}}{=} & \mathbf{0}\\
der\,c\,(\mathbf{1}) & \stackrel{\text{def}}{=} & \mathbf{0}\\
der\,c\,(d) & \stackrel{\text{def}}{=} & \text{if } c = d \text{ then } \mathbf{1} \text{ else } \mathbf{0}\\
der\,c\,(r_1 + r_2) & \stackrel{\text{def}}{=} & der\,c\,r_1 + der\,c\,r_2\\
der\,c\,(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & \text{if } nullable(r_1)\\
 & & \text{then } (der\,c\,r_1) \cdot r_2 + der\,c\,r_2\\
 & & \text{else } (der\,c\,r_1) \cdot r_2\\
der\,c\,(r^*) & \stackrel{\text{def}}{=} & (der\,c\,r) \cdot (r^*)\\
ders\,[]\,r & \stackrel{\text{def}}{=} & r\\
ders\,(c::s)\,r & \stackrel{\text{def}}{=} & ders\,s\,(der\,c\,r)
\end{array}
$$

Example

Given $r \stackrel{\text{def}}{=} ((a \cdot b) + b)^*$ what is

$$
\begin{array}{rcl}
der\,a\,((a \cdot b) + b)^* & \Rightarrow & der\,a\,\underline{((a \cdot b) + b)^*} \\
& = & (der\,a\,(\underline{(a \cdot b) + b})) \cdot r \\
& = & ((der\,a\,(\underline{a \cdot b})) + (der\,a\,b)) \cdot r \\
& = & (((der\,a\,\underline{a}) \cdot b) + (der\,a\,b)) \cdot r \\
& = & ((\mathbf{1} \cdot b) + (der\,a\,\underline{b})) \cdot r \\
& = & ((\mathbf{1} \cdot b) + \mathbf{0}) \cdot r
\end{array}
$$

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

Simplification

Given $r \stackrel{\text{def}}{=} ((a \cdot b) + b)^*$, you can simplify as follows

$$\begin{aligned}
((\mathbf{1} \cdot b) + \mathbf{0}) \cdot r &\Rightarrow ((\underline{\mathbf{1} \cdot b}) + \mathbf{0}) \cdot r \\
&= (\underline{b + \mathbf{0}}) \cdot r \\
&= b \cdot r
\end{aligned}$$

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.

We proved

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

by induction on the regular expression r.

We proved

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

by induction on the regular expression r.

Any Questions?

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

Correctness Proof for our Matcher

- We started from

$$\begin{array}{ll} & s \in L(r) \\ \Leftrightarrow & [] \in Ders\, s\, (L(r)) \end{array}$$

Correctness Proof for our Matcher

- We started from

$$\begin{array}{ll} & s \in L(r) \\ \Leftrightarrow & [] \in Ders\, s\, (L(r)) \end{array}$$

- **if** we can show $Ders\, s\, (L(r)) = L(ders\, s\, r)$ we have

$$\begin{array}{ll} \Leftrightarrow & [] \in L(ders\, s\, r) \\ \Leftrightarrow & nullable(ders\, s\, r) \\ \stackrel{\text{def}}{=} & matches\, s\, r \end{array}$$

We need to prove

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

also by induction on the regular expression r.

(Basic) Regular Expressions

r	$::=$	$\mathbf{0}$	nothing
	$\mid$	$\mathbf{1}$	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

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!

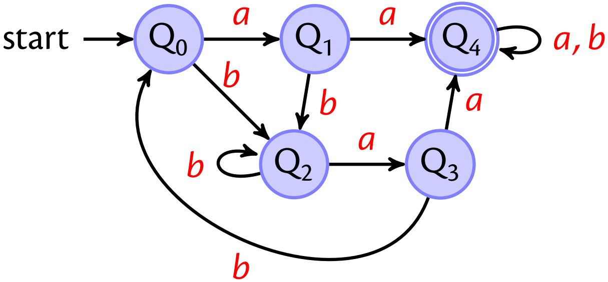
Automata

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

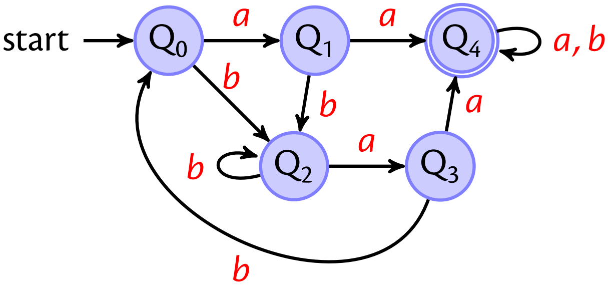
- an alphabet Σ
- a set of states Qs
- one of these states is the start state Q_0
- some states are accepting states F, and
- there is transition function δ

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

$$A(\Sigma, Qs, Q_0, F, \delta)$$



- 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

$$A(\Sigma, Qs, Q_0, F, \delta)$$

you can define

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

Accepting a String

Given

$$A(\Sigma, \mathrm{Qs}, \mathrm{Q}_0, F, \delta)$$

you can define

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

Whether a string s is accepted by A?

$$\widehat{\delta}(\mathrm{Q}_0, s) \in F$$

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 (NFA) consists again of:

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

$$\begin{array}{l} (Q_1, a) \rightarrow Q_2 \\ (Q_1, a) \rightarrow Q_3 \end{array} \quad \ldots$$

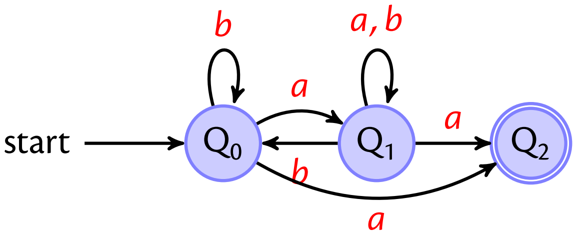
Non-Deterministic Finite Automata

A non-deterministic finite automaton (NFA) consists again of:

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

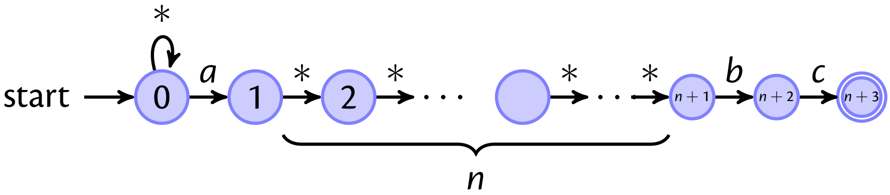
$$(Q_1, a) \rightarrow Q_2 \quad (Q_1, a) \rightarrow Q_3 \quad \ldots \qquad (Q_1, a) \rightarrow \{Q_2, Q_3\}$$

An NFA Example



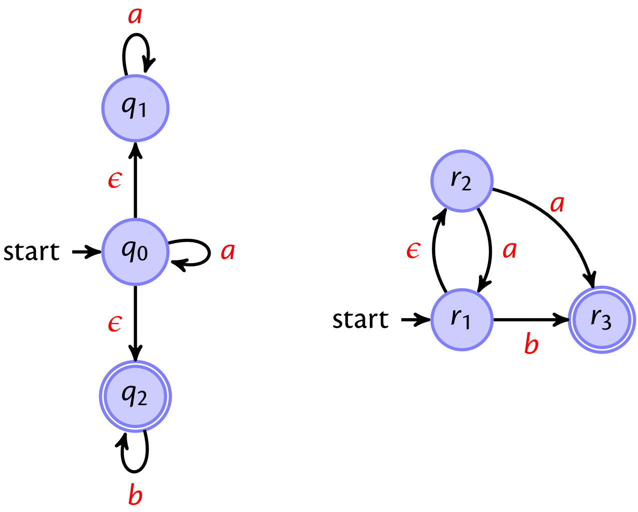
Another Example

For the regular expression $(.^*)a\,(.^{\{n\}})bc$



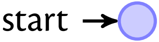
Note the star-transitions: accept any character.

Two Epsilon NFA Examples

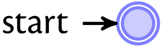


Rexp to NFA

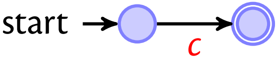
$\mathbf{0}$



$\mathbf{1}$

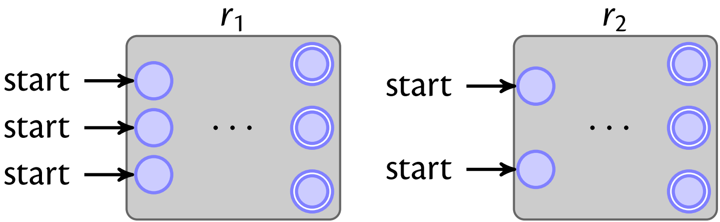


c



Case $r_1 \cdot r_2$

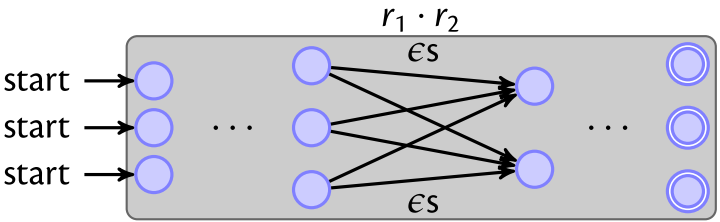
By recursion we are given two automata:



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 $r_1 \cdot r_2$

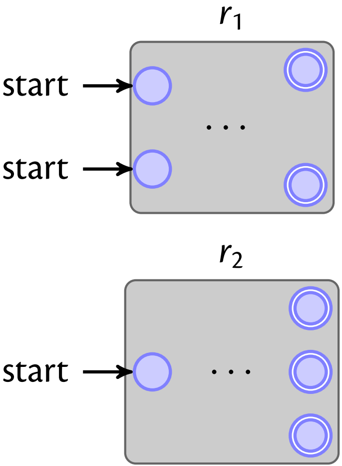
By recursion we are given two automata:



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 $r_1 + r_2$

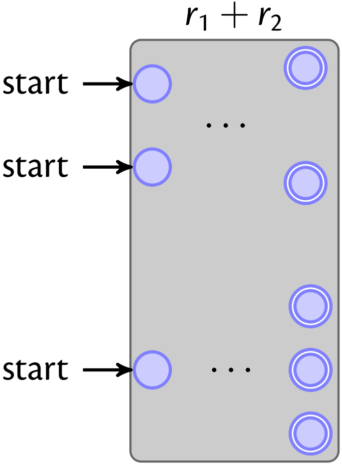
By recursion we are given two automata:



We can just put both automata together.

Case $r_1 + r_2$

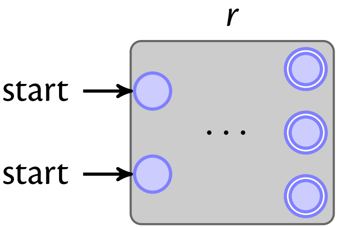
By recursion we are given two automata:



We can just put both automata together.

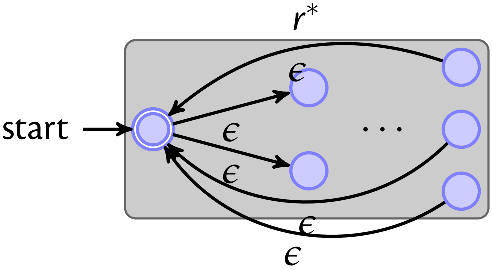
Case r^*

By recursion we are given an automaton for r:



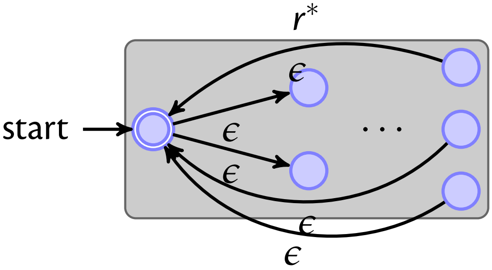
Case r^*

By recursion we are given an automaton for r:



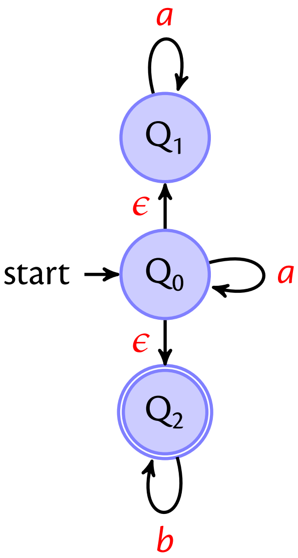
Case r^*

By recursion we are given an automaton for r:



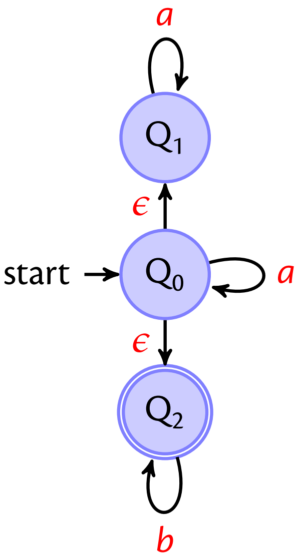
Why can't we just have an epsilon transition from the accepting states to the starting state?

Subset Construction



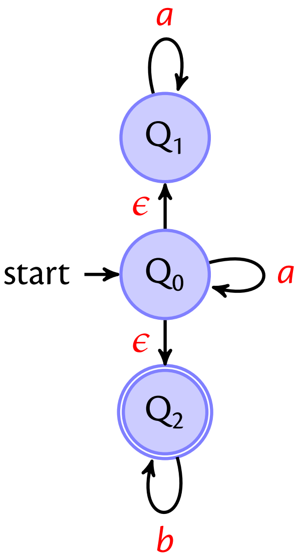
nodes	a	b
$\{\}$		
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



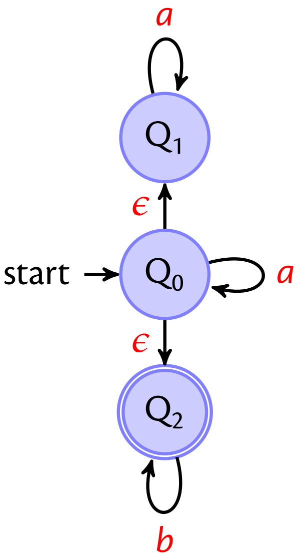
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



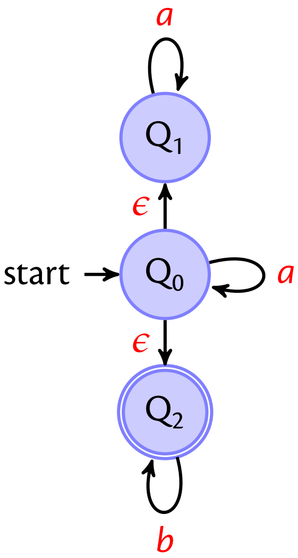
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



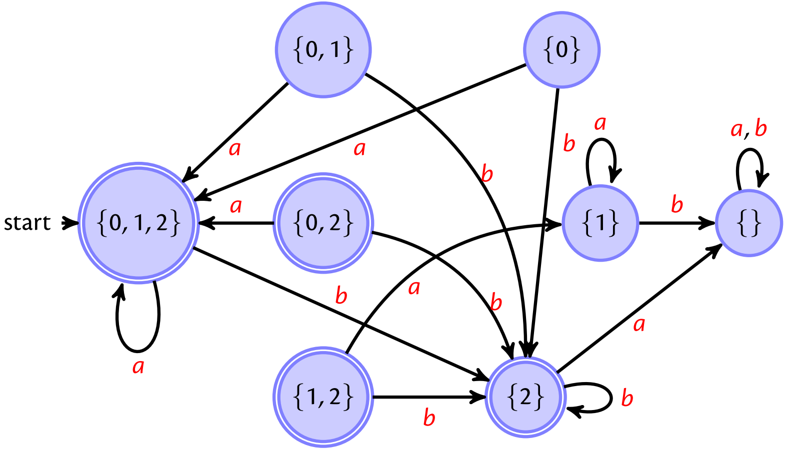
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0,1\}$	$\{0,1,2\}$	$\{2\}$
$\{0,2\}$	$\{0,1,2\}$	$\{2\}$
$\{1,2\}$	$\{1\}$	$\{2\}$
$\{0,1,2\}$	$\{0,1,2\}$	$\{2\}$

Subset Construction



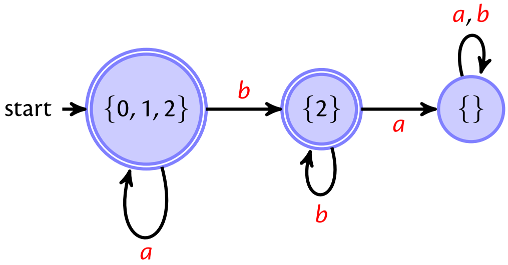
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$ *	$\{\}$	$\{2\}$
$\{0,1\}$	$\{0,1,2\}$	$\{2\}$
$\{0,2\}$ *	$\{0,1,2\}$	$\{2\}$
$\{1,2\}$ *	$\{1\}$	$\{2\}$
s: $\{0,1,2\}$ *	$\{0,1,2\}$	$\{2\}$

The Result

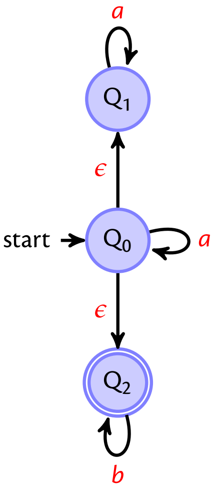


Removing Dead States

DFA:



(original) NFA:

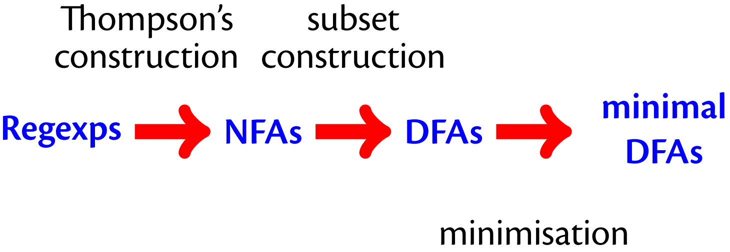


Regexps and Automata

Thompson's construction	subset construction

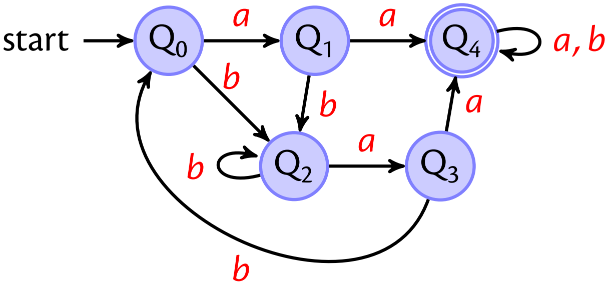
Regexps → **NFAs** → **DFAs**

Regexps and Automata

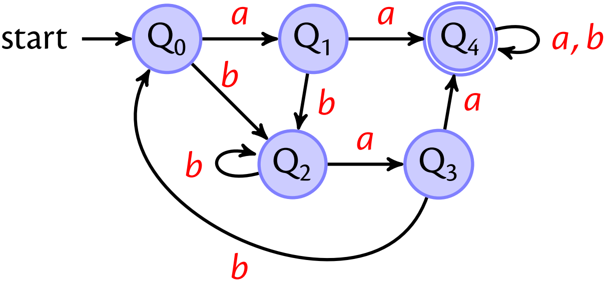


DFA Minimisation

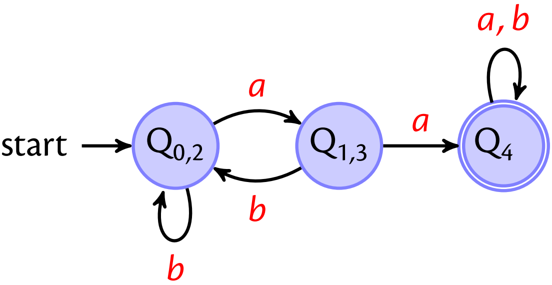
1. Take all pairs (q, p) with $q \neq p$
2. Mark all pairs that accepting and non-accepting states
3. For all unmarked pairs (q, p) and all characters c test whether
   $$(\delta(q, c), \delta(p, c))$$
   are marked. If yes in at least one case, then also mark (q, p).
4. Repeat last step until no change.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



	Q_0	Q_1	Q_2	Q_3
Q_1				
Q_2				
Q_3				
Q_4	$\star$	$\star$	$\star$	$\star$

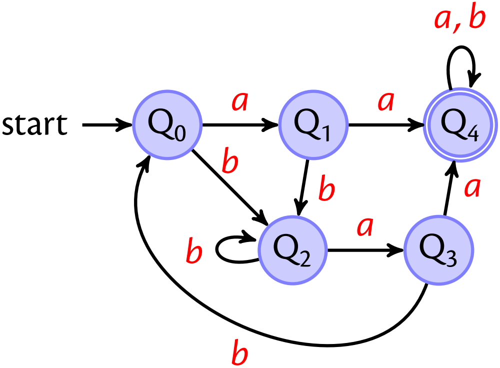


	Q_0	Q_1	Q_2	Q_3
Q_1	★			
Q_2		★		
Q_3	★		★	
Q_4	★	★	★	★



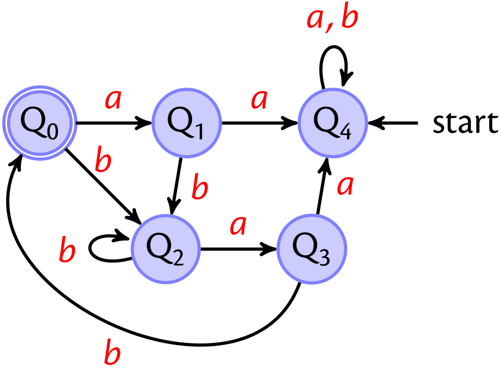
minimal automaton

Alternatives



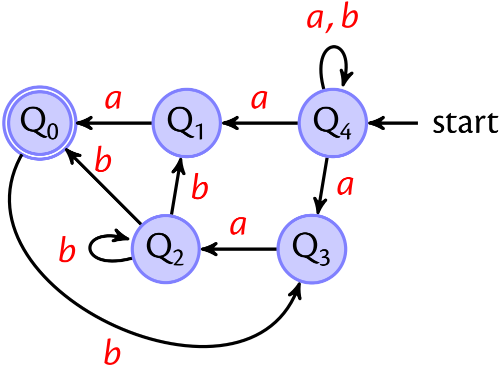
- exchange initial / accepting states

Alternatives



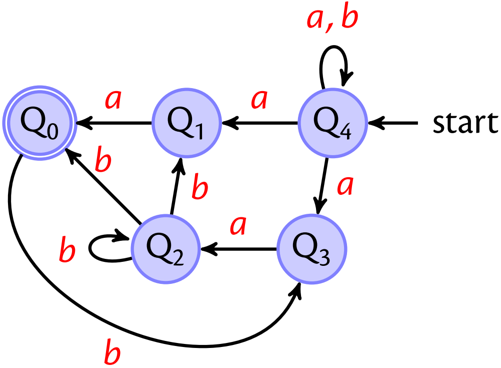
- exchange initial / accepting states
- reverse all edges

Alternatives



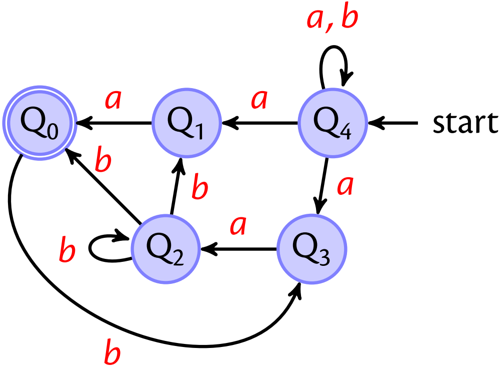
- exchange initial / accepting states
- reverse all edges
- subset construction $\Rightarrow$ DFA

Alternatives



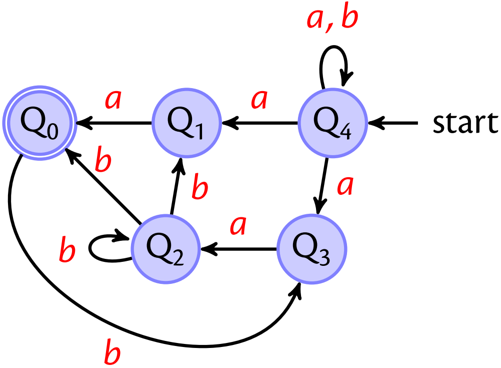
- exchange initial / accepting states
- reverse all edges
- subset construction ⇒ DFA
- remove dead states

Alternatives



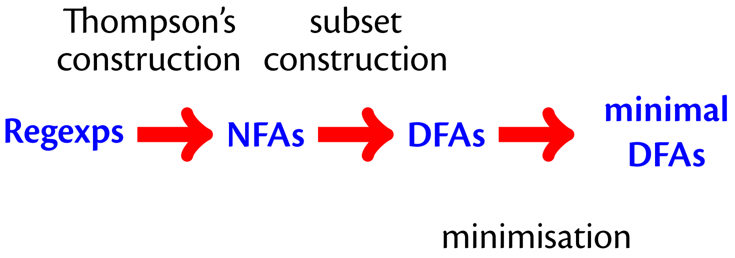
- exchange initial / accepting states
- reverse all edges
- subset construction ⇒ DFA
- remove dead states
- repeat once more

Alternatives

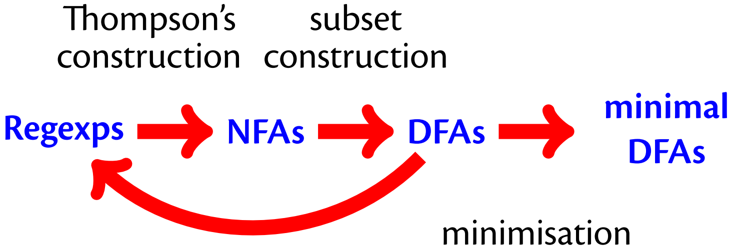


- exchange initial / accepting states
- reverse all edges
- subset construction ⇒ DFA
- remove dead states
- repeat once more ⇒ minimal DFA

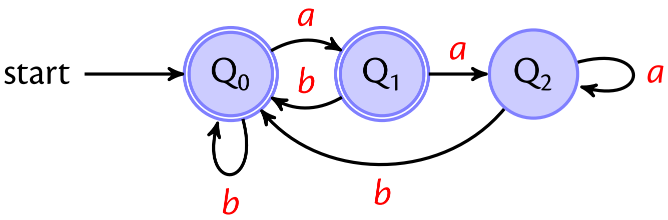
Regexps and Automata

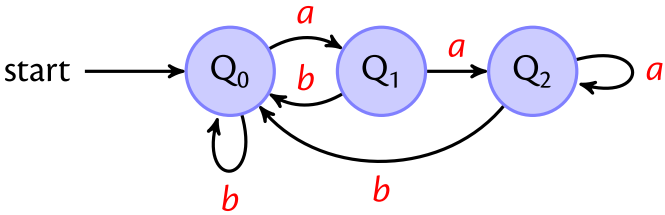


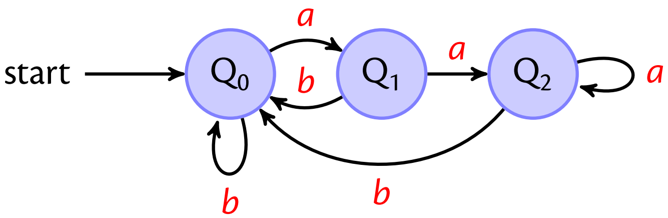
Regexps and Automata



DFA to Rexp

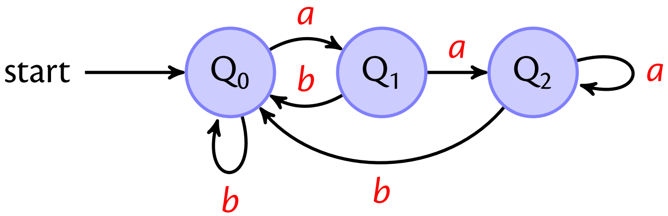


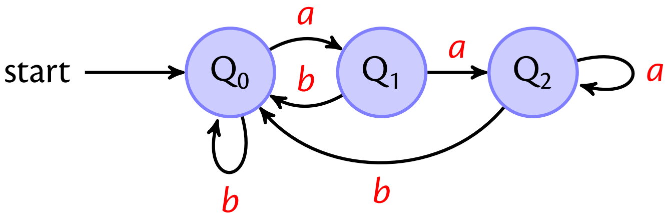

start
Q0
Q1
Q2
a
b
a
a
b
b



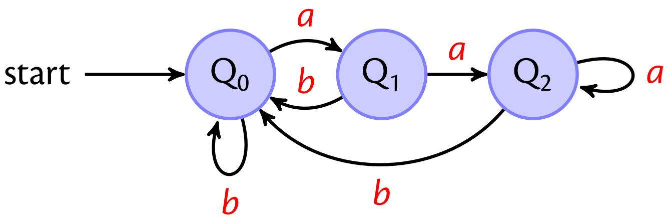
You know how to solve since school days, no?

$$
\begin{aligned}
Q_0 &= 2\,Q_0 + 3\,Q_1 + 4\,Q_2 \\
Q_1 &= 2\,Q_0 + 3\,Q_1 + 1\,Q_2 \\
Q_2 &= 1\,Q_0 + 5\,Q_1 + 2\,Q_2
\end{aligned}
$$


start
Q_0
Q_1
Q_2
a
a
a
b
b
b



$$\begin{aligned} Q_0 &= Q_0\, b + Q_1\, b + Q_2\, b + \mathbf{1} \\ Q_1 &= Q_0\, a \\ Q_2 &= Q_1\, a + Q_2\, a \end{aligned}$$

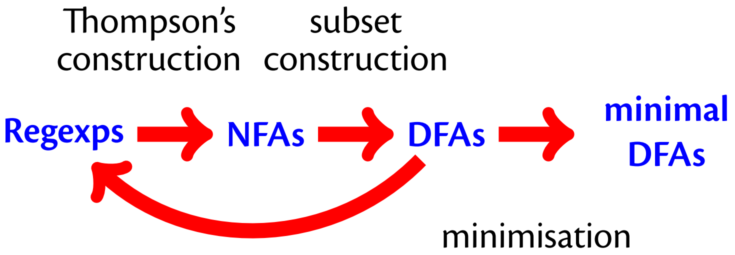


$$
\begin{aligned}
Q_0 &= Q_0\,b + Q_1\,b + Q_2\,b + \mathbf{1}\\
Q_1 &= Q_0\,a\\
Q_2 &= Q_1\,a + Q_2\,a
\end{aligned}
$$

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

$$\begin{aligned}
rev(\mathbf{0}) &\stackrel{\text{def}}{=} \mathbf{0}\\
rev(\mathbf{1}) &\stackrel{\text{def}}{=} \mathbf{1}\\
rev(c) &\stackrel{\text{def}}{=} c\\
rev(r_1 + r_2) &\stackrel{\text{def}}{=} rev(r_1) + rev(r_2)\\
rev(r_1 \cdot r_2) &\stackrel{\text{def}}{=} rev(r_2) \cdot rev(r_1)\\
rev(r^*) &\stackrel{\text{def}}{=} rev(r)^*
\end{aligned}$$

and the set

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

prove whether

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