Compilers and Formal Languages

Email:	christian.urban at kcl.ac.uk
Office Hour:	Friday 12 – 14
Location:	N7.07 (North Wing, Bush House)
Slides & Progs:	KEATS
Pollev:	`https://pollev.com/cfltutoratki576`

For Installation Problems

- Harry Dilnot (harry.dilnot@kcl.ac.uk)
  Windows expert
- Oliver Iliffe (oliver.iliffe@kcl.ac.uk)

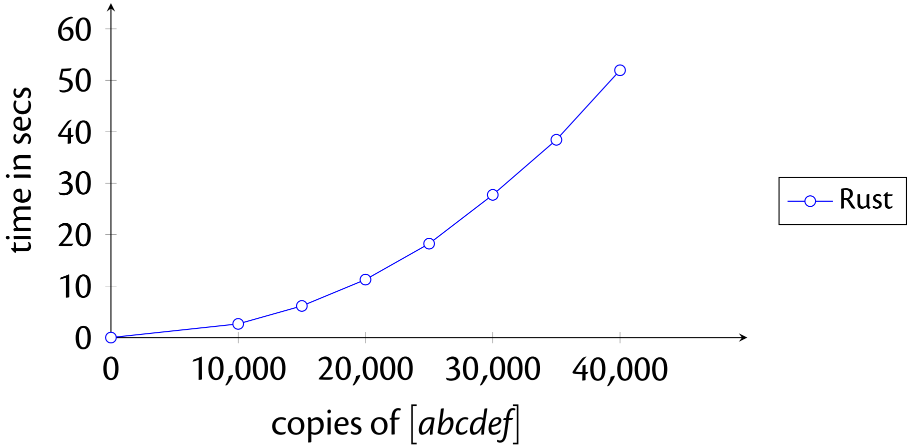
From Pollev last week

Is the equivalence of two regexes belong in the P or NP class of problems?

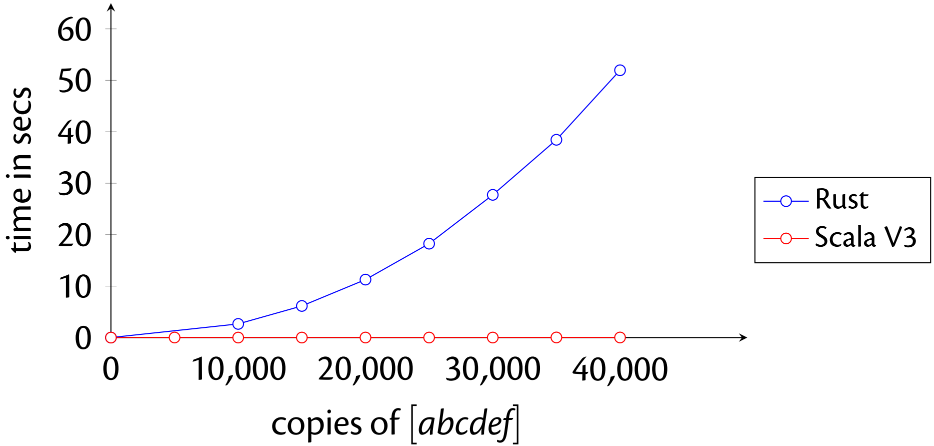
From Pollev last week

If state machines are not efficient, then how/why do many lexer packages like the logos crate in rust compile down a lexer definition down to a jump table driven state machine? Could we achieve quicker lexing with things like SIMD instructions?

$(abcdef)^{\{n\}}$ in Rust and Scala



$(abcdef)^{\{n\}}$ in Rust and Scala



From Pollev last week

For a regular expression $r = r_1 \cdot r_2$, *to prove that* $\textit{der}\ c\ r = (\textit{der}\ c\ r) \cdot r^{\{n-1\}}$, *is there a way to prove it in the general case instead of how you do the calculations for each* n *in the videos?*

(Basic) Regular Expressions

$$
\begin{array}{rcll}
r & ::= & \mathbf{0} & \text{nothing} \\
  & \mid & \mathbf{1} & \text{empty string / "" / []} \\
  & \mid & c & \text{character} \\
  & \mid & r_1 \cdot r_2 & \text{sequence} \\
  & \mid & r_1 + r_2 & \text{alternative / choice} \\
  & \mid & r^* & \text{star (zero or more)}
\end{array}
$$

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!

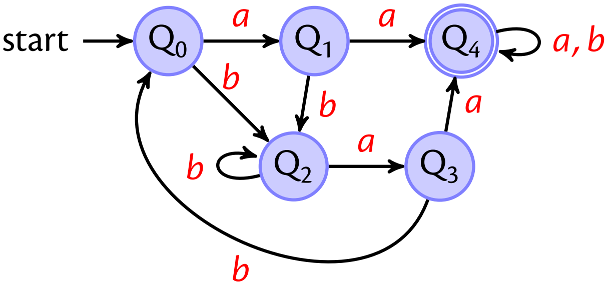
Automata

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

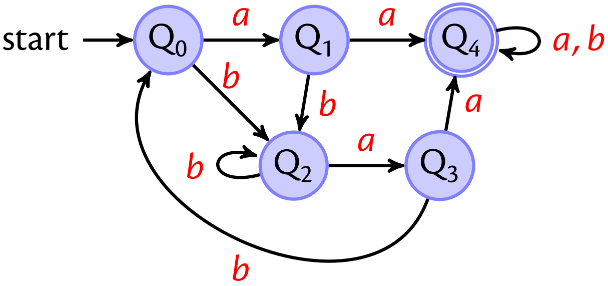
- an alphabet Σ
- a set of states Qs
- one of these states is the start state Q_0
- some states are accepting states F, and
- there is transition function δ

  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

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

Accepting a String

Given

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

you can define

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

Whether a string s is accepted by A?

$$\widehat{\delta}(Q_0, s) \in F$$

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

$N(\Sigma, Qs, Qs_0, F, \rho)$

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

- a finite set of states, Qs
- some these states are the start states, Qs_0
- some states are accepting states, and
- there is transition relation, ρ

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

Non-Deterministic Finite Automata

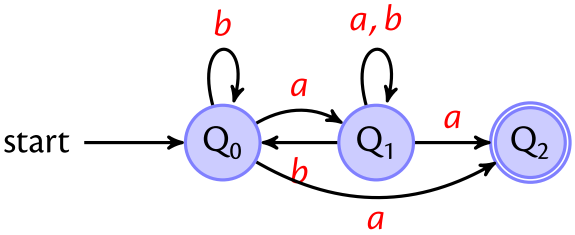
$N(\Sigma, Qs, Qs_0, F, \rho)$

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

- a finite set of states, Qs
- some these states are the start states, Qs_0
- some states are accepting states, and
- there is transition relation, ρ

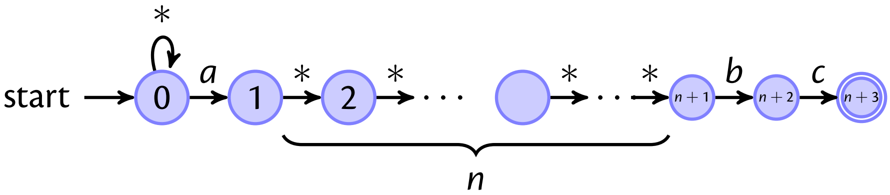
$$\begin{array}{l} (Q_1, a) \rightarrow Q_2 \\ (Q_1, a) \rightarrow Q_3 \end{array} \quad \ldots \qquad\qquad (Q_1, a) \rightarrow \{Q_2, Q_3\}$$

An NFA Example



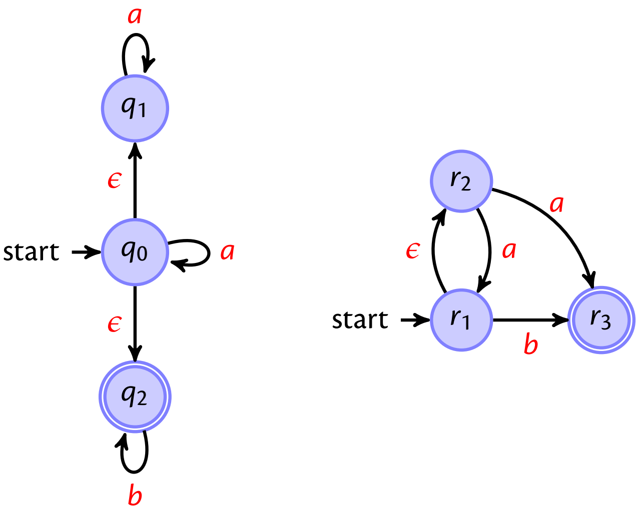
Another Example

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



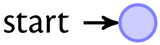
Note the star-transitions: accept any character.

Two Epsilon NFA Examples

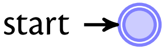


Thompson: Rexp to ϵNFA

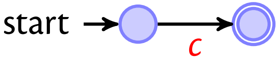
$\mathbf{0}$



$\mathbf{1}$

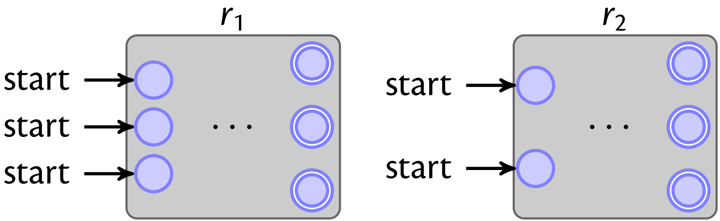


c



Case $r_1 \cdot r_2$

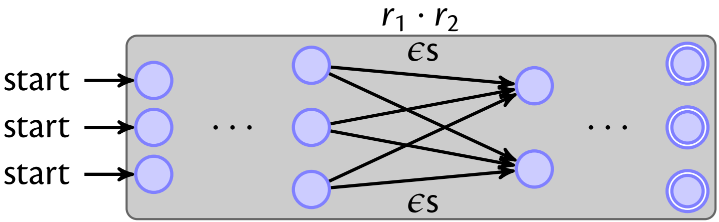
By recursion we are given two automata:



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 $r_1 \cdot r_2$

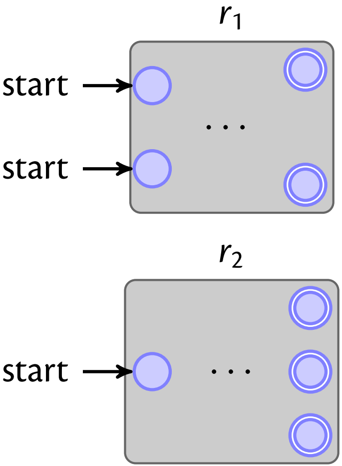
By recursion we are given two automata:



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 $r_1 + r_2$

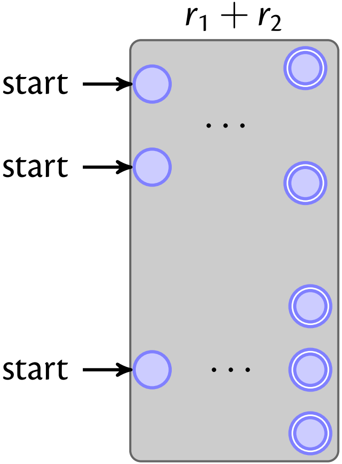
By recursion we are given two automata:



We can just put both automata together.

Case $r_1 + r_2$

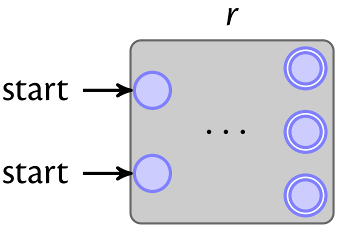
By recursion we are given two automata:



We can just put both automata together.

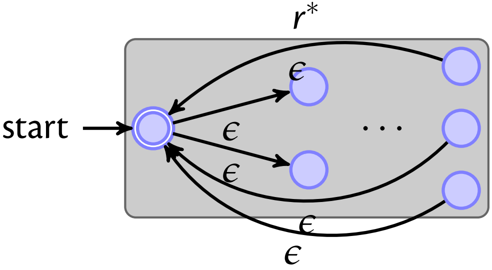
Case r^*

By recursion we are given an automaton for r:



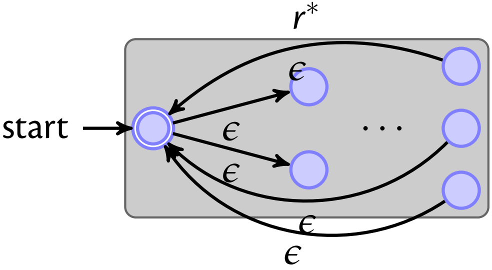
Case r^*

By recursion we are given an automaton for r:



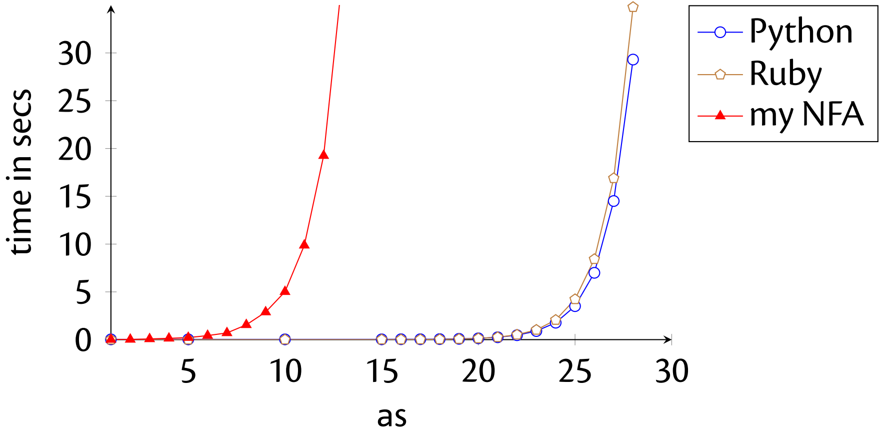
Case r^*

By recursion we are given an automaton for r:

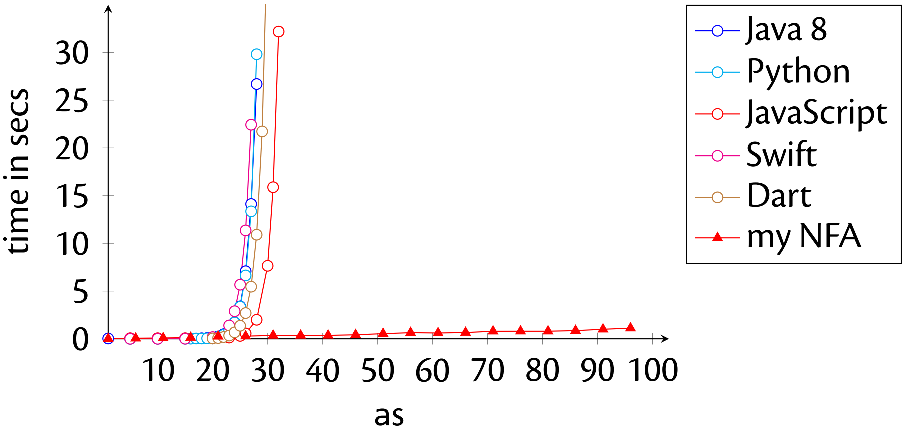


Why can't we just have an epsilon transition from the accepting states to the starting state?

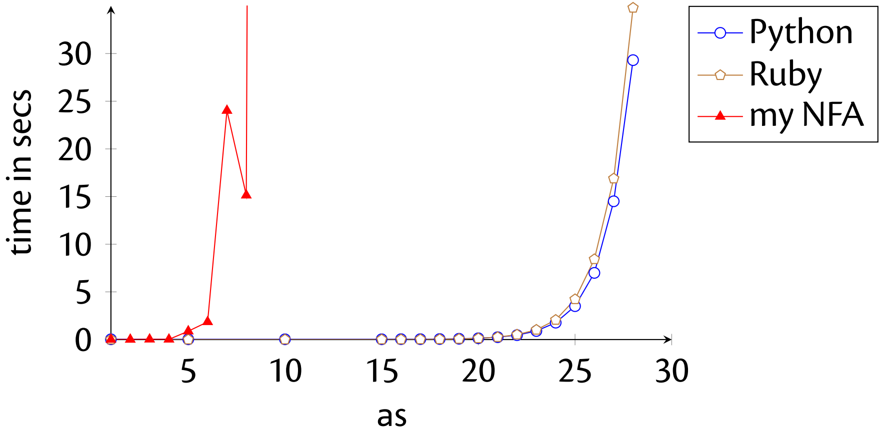
NFA Breadth-First: $a^{?\{n\}} \cdot a^{\{n\}}$



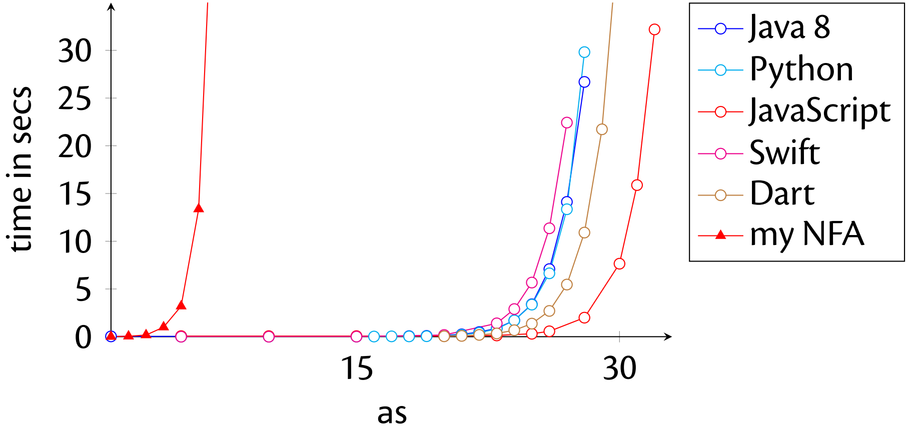
NFA Breadth-First: $(a^*)^* \cdot b$



NFA Depth-First: $a^{?\{n\}} \cdot a^{\{n\}}$

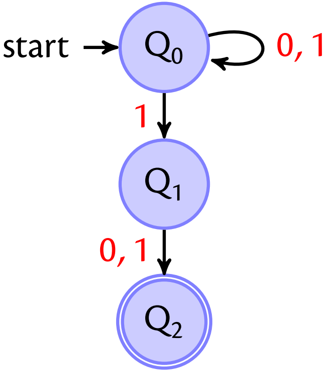


NFA Depth-First: $(a^*)^* \cdot b$



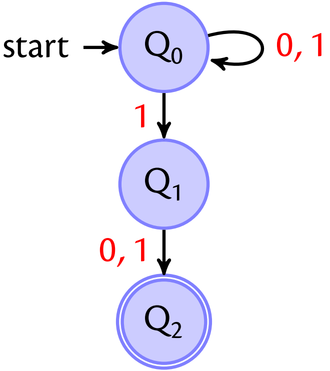
The punchline is that many existing libraries do depth-first search in NFAs (with backtracking).

Subset Construction



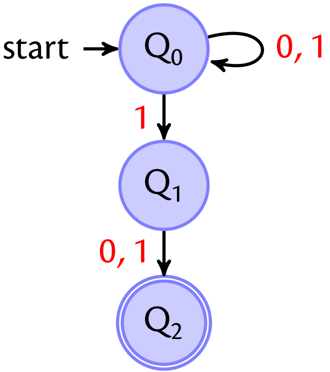
nodes	0	1
$\{\}$		
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



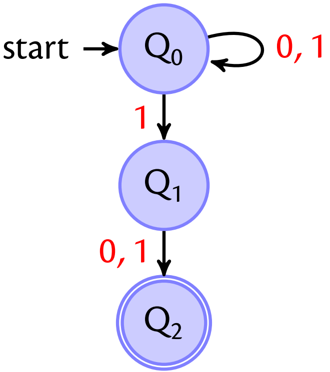
nodes	0	1
$\{\}$	$\{\}$	$\{\}$
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

Subset Construction



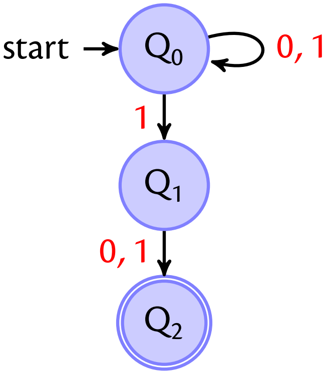
nodes	0	1
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0\}$	$\{0,1\}$
$\{1\}$	$\{2\}$	$\{2\}$
$\{2\}$	$\{\}$	$\{\}$
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



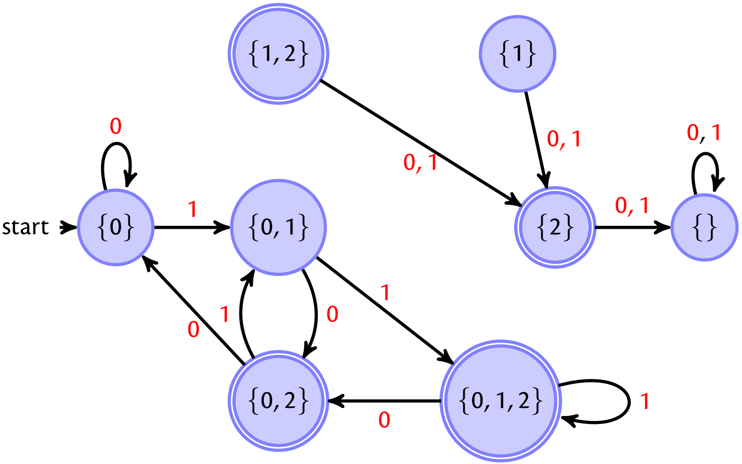
nodes	0	1
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0\}$	$\{0,1\}$
$\{1\}$	$\{2\}$	$\{2\}$
$\{2\}$	$\{\}$	$\{\}$
$\{0,1\}$	$\{0,2\}$	$\{0,1,2\}$
$\{0,2\}$	$\{0\}$	$\{0,1\}$
$\{1,2\}$	$\{2\}$	$\{2\}$
$\{0,1,2\}$	$\{0,2\}$	$\{0,1,2\}$

Subset Construction



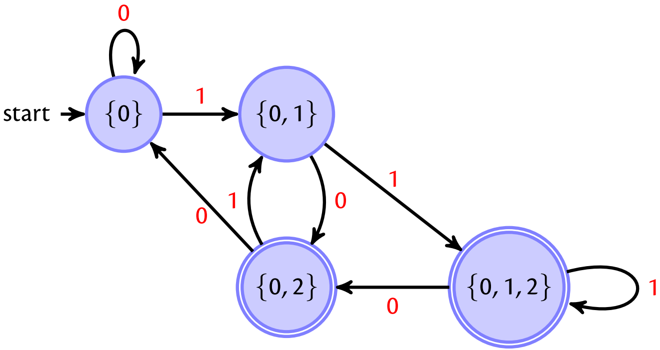
nodes	0	1
$\{\}$	$\{\}$	$\{\}$
s: $\{0\}$	$\{0\}$	$\{0,1\}$
$\{1\}$	$\{2\}$	$\{2\}$
$\{2\}$ *	$\{\}$	$\{\}$
$\{0,1\}$	$\{0,2\}$	$\{0,1,2\}$
$\{0,2\}$ *	$\{0\}$	$\{0,1\}$
$\{1,2\}$ *	$\{2\}$	$\{2\}$
$\{0,1,2\}$ *	$\{0,2\}$	$\{0,1,2\}$

The Result

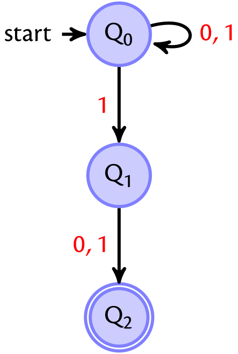


Removing Dead States

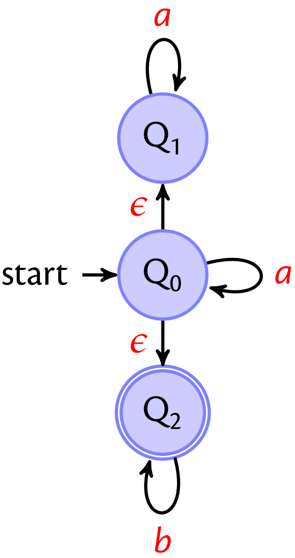
DFA:



(original) NFA:

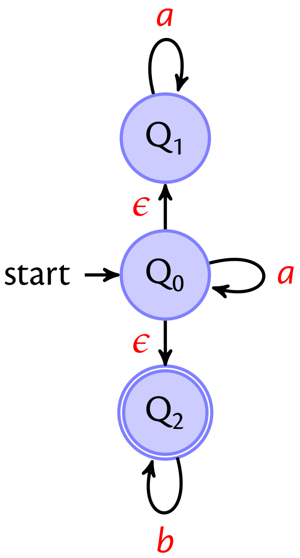


Subset Construction (ϵNFA)



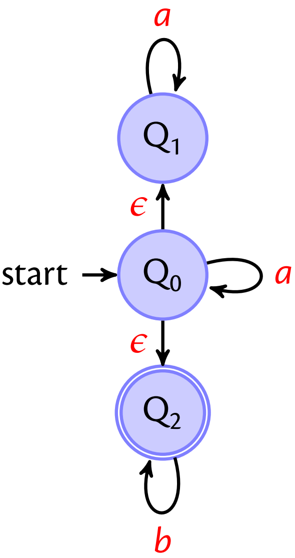
nodes	a	b
$\{\}$		
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction (ϵNFA)



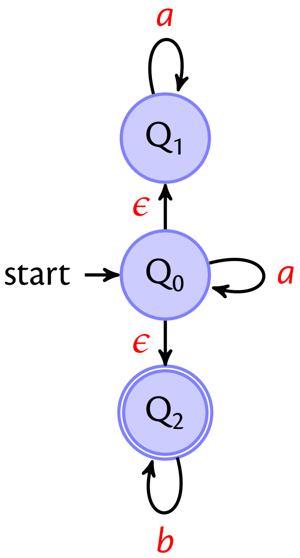
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction (ϵNFA)



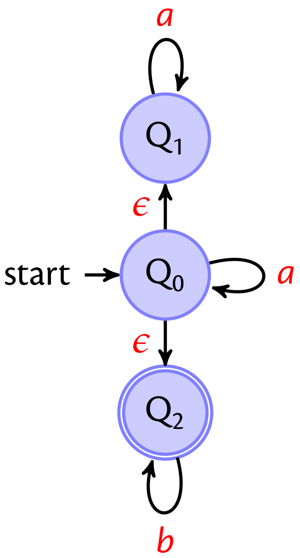
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction (ϵNFA)



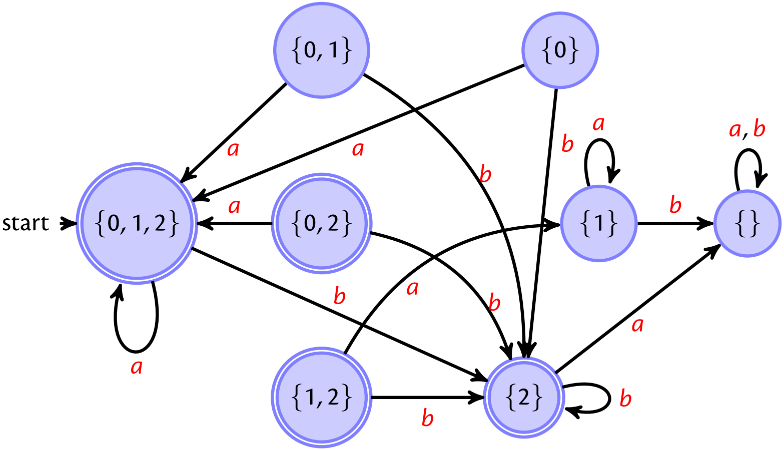
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0,1\}$	$\{0,1,2\}$	$\{2\}$
$\{0,2\}$	$\{0,1,2\}$	$\{2\}$
$\{1,2\}$	$\{1\}$	$\{2\}$
$\{0,1,2\}$	$\{0,1,2\}$	$\{2\}$

Subset Construction (ϵNFA)



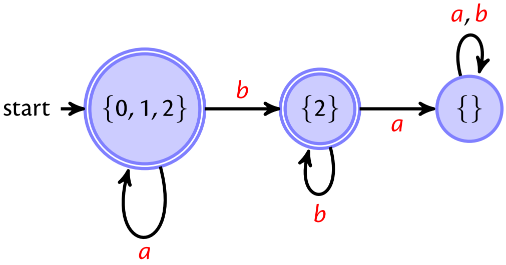
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$ *	$\{\}$	$\{2\}$
$\{0,1\}$	$\{0,1,2\}$	$\{2\}$
$\{0,2\}$ *	$\{0,1,2\}$	$\{2\}$
$\{1,2\}$ *	$\{1\}$	$\{2\}$
s: $\{0,1,2\}$ *	$\{0,1,2\}$	$\{2\}$

The Result

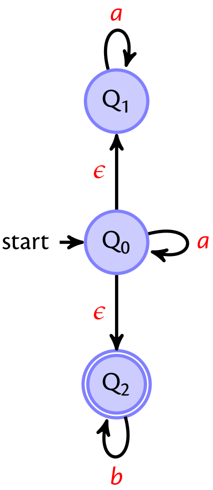


Removing Dead States

DFA:



(original) NFA:



Regexps and Automata

Thompson's construction	subset construction

Regexps $\rightarrow$ **NFAs** $\rightarrow$ **DFAs**

Regexps and Automata

Thompson's construction

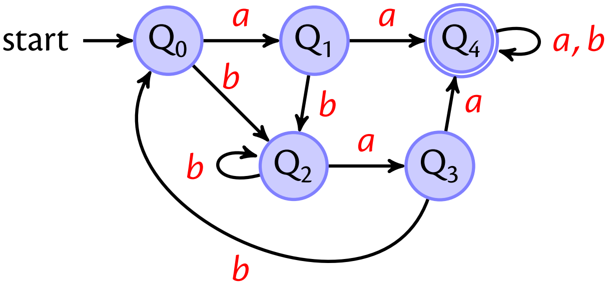
subset construction

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

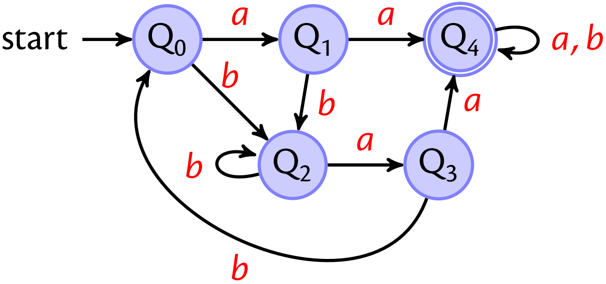
minimisation

DFA Minimisation

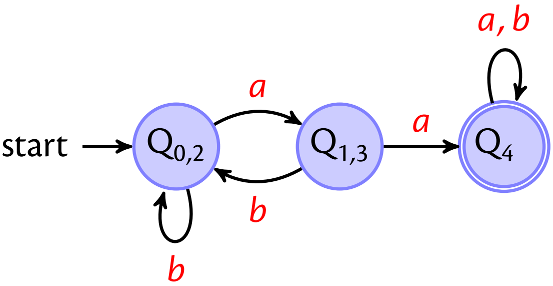
1. Take all pairs (q, p) with $q \neq p$
2. Mark all pairs that accepting and non-accepting states
3. For all unmarked pairs (q, p) and all characters c test whether
$$(\delta(q, c), \delta(p, c))$$
are marked. If yes in at least one case, then also mark (q, p).
4. Repeat last step until no change.
5.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



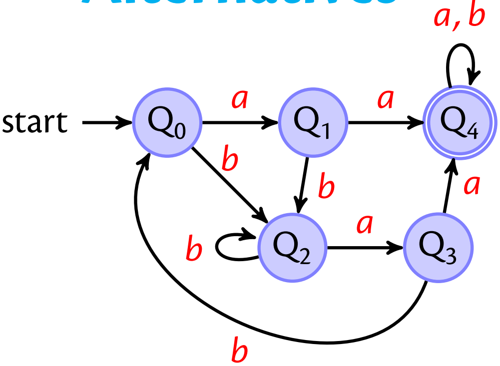
	Q_0	Q_1	Q_2	Q_3
Q_1				
Q_2				
Q_3				
Q_4	$\star$	$\star$	$\star$	$\star$



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

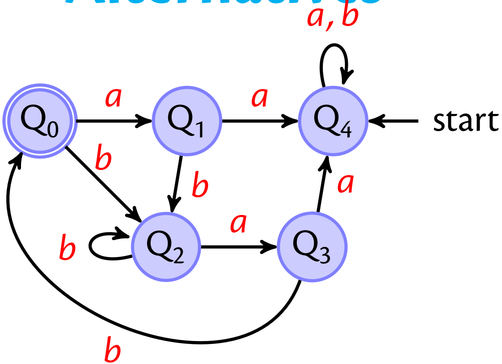


Alternatives



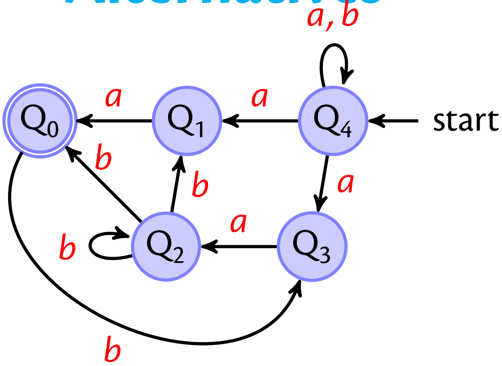
- exchange initial / accepting states

Alternatives



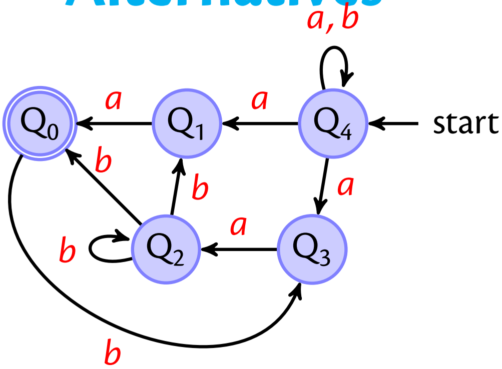
- exchange initial / accepting states
- reverse all edges

Alternatives



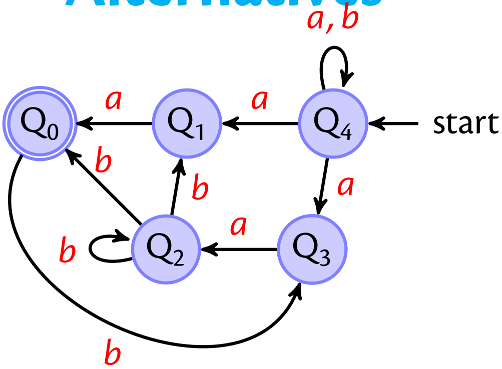
- exchange initial / accepting states
- reverse all edges
- subset construction $\Rightarrow$ DFA

Alternatives



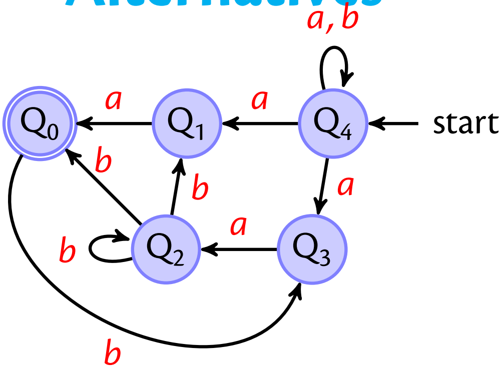
- exchange initial / accepting states
- reverse all edges
- subset construction $\Rightarrow$ DFA
- remove dead states

Alternatives



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

Alternatives



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

Regexps and Automata

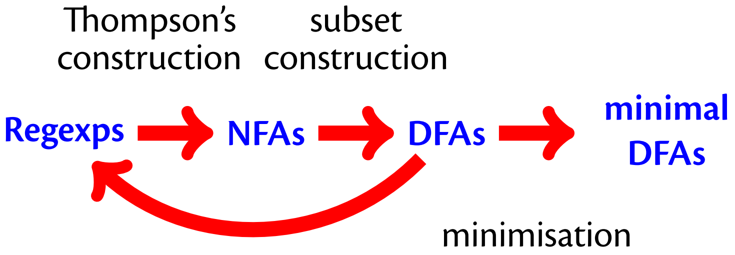
Thompson's construction

subset construction

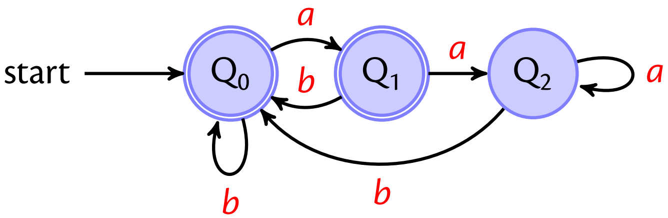
Regexps → NFAs → DFAs → minimal DFAs

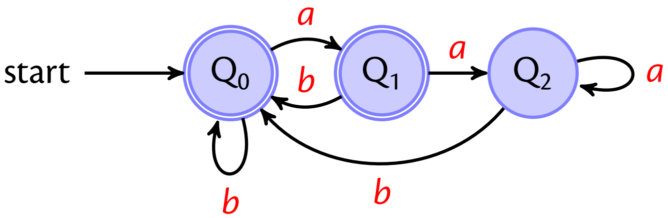
minimisation

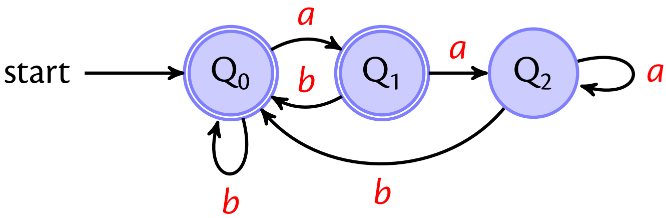
Regexps and Automata



DFA to Rexp

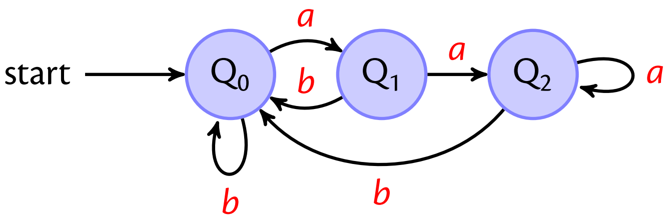


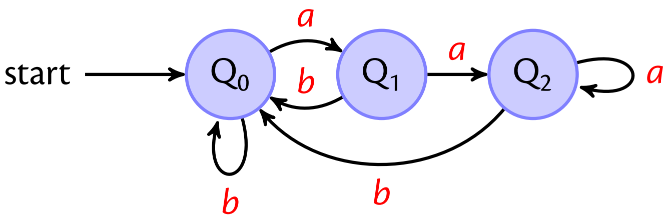

start
Q_0
Q_1
Q_2
a
a
a
b
b
b



You know how to solve since school days, no?

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

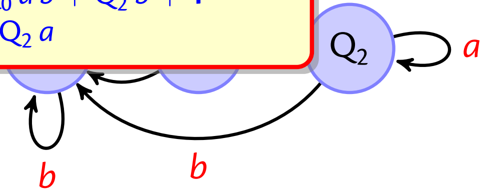

start
Q0
Q1
Q2
a
b
a
a
b
b



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

substitute Q_1 into Q_0 & Q_2:

$$Q_0 = Q_0\, b + Q_0\, a\, b + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a + Q_2\, a$$



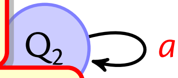
$$
\begin{aligned}
Q_0 &= Q_0\, b + Q_1\, b + Q_2\, b + \mathbf{1}\\
Q_1 &= Q_0\, a\\
Q_2 &= Q_1\, a + Q_2\, a
\end{aligned}
$$

substitute Q_1 into Q_0 & Q_2:

$$Q_0 = Q_0\,b + Q_0\,a\,b + Q_2\,b + \mathbf{1}$$
$$Q_2 = Q_0\,a\,a + Q_2\,a$$

simplifying Q_0:

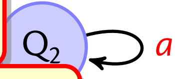
$$Q_0 = Q_0\,(b + a\,b) + Q_2\,b + \mathbf{1}$$
$$Q_2 = Q_0\,a\,a + Q_2\,a$$



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

substitute Q_1 into Q_0 & Q_2:

$$Q_0 = Q_0\,b + Q_0\,a\,b + Q_2\,b + \mathbf{1}$$
$$Q_2 = Q_0\,a\,a + Q_2\,a$$



simplifying Q_0:

$$Q_0 = Q_0\,(b + a\,b) + Q_2\,b + \mathbf{1}$$
$$Q_2 = Q_0\,a\,a + Q_2\,a$$

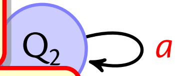
$$\begin{aligned} Q_0 &= Q_0\,b + Q_1\,b + Q_2\,b + \mathbf{1} \\ Q_1 &= Q_0\,a \end{aligned}$$

Arden's Lemma:

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

substitute Q_1 into Q_0 & Q_2:

$$Q_0 = Q_0\, b + Q_0\, a\, b + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a + Q_2\, a$$



simplifying Q_0:

$$Q_0 = Q_0\,(b + a\, b) + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a + Q_2\, a$$

Arden for Q_2:

$$Q_0 = Q_0\,(b + a\, b) + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a\,(a^*)$$

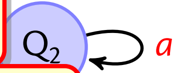
$$Q_1 = Q_0\, a$$

Arden's Lemma:

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

substitute Q_1 into Q_0 & Q_2:

$$Q_0 = Q_0\, b + Q_0\, a\, b + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a + Q_2\, a$$



simplifying Q_0:

$$Q_0 = Q_0\, (b + a\, b) + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a + Q_2\, a$$

Arden for Q_2:

$$Q_0 = Q_0\, (b + a\, b) + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a\, (a^*)$$

$Q_1 =$

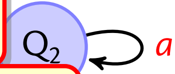
$Q_2 =$

Substitute Q_2 and simplify:

$$Q_0 = Q_0\, (b + a\, b + a\, a\, (a^*)\, b) + \mathbf{1}$$

substitute Q_1 into Q_0 & Q_2:

$$Q_0 = Q_0\, b + Q_0\, a\, b + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a + Q_2\, a$$



simplifying Q_0:

$$Q_0 = Q_0\, (b + a\, b) + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a + Q_2\, a$$

Arden's Lemma:

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

$\mathbf{1}$

$Q_1 =$

$Q_2 =$

Substitute Q_2 and simplify:

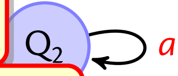
$$Q_0 = Q_0\, (b + a\, b + a\, a\, (a^*)\, b) + \mathbf{1}$$

Arden again for Q_0:

$$Q_0 = (b + a\, b + a\, a\, (a^*)\, b)^*$$

substitute Q_1 into Q_0 & Q_2:

$$Q_0 = Q_0\, b + Q_0\, a\, b + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a + Q_2\, a$$



simplifying Q_0:

$$Q_0 = Q_0\, (b + a\, b) + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a + Q_2\, a$$

Arden for Q_2:

$$Q_0 = Q_0\, (b + a\, b) + Q_2\, b + \mathbf{1}$$
$$Q_2 = Q_0\, a\, a\, (a^*)$$

$Q_1 =$

$Q_2 =$

Substitute

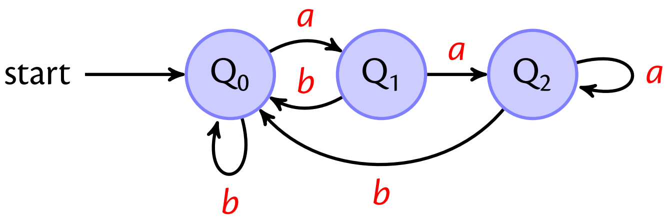
$Q_0 = Q_0$

Arde

Q_0

Finally:

$$Q_0 = (b + a\, b + a\, a\, (a^*)\, b)^*$$
$$Q_1 = (b + a\, b + a\, a\, (a^*)\, b)^*\, a$$
$$Q_2 = (b + a\, b + a\, a\, (a^*)\, b)^*\, a\, a\, (a^*)$$

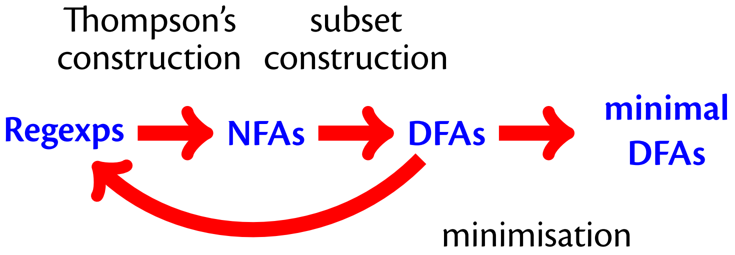


$$Q_0 = Q_0\, b + Q_1\, b + Q_2\, b + \mathbf{1}$$
$$Q_1 = Q_0\, a$$
$$Q_2 = Q_1\, a + Q$$

Finally:

$$Q_0 = (b + a\, b + a\, a\, (a^*)\, b)^*$$
$$Q_1 = (b + a\, b + a\, a\, (a^*)\, b)^*\, a$$
$$Q_2 = (b + a\, b + a\, a\, (a^*)\, b)^*\, a\, a\, (a^*)$$

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)

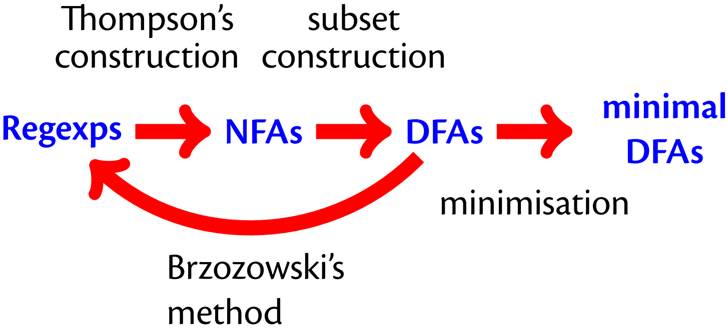
A language is **regular** iff there exists a regular expression that recognises all its strings.

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?

Regexps and Automata



Regular Languages

Two equivalent definitions:

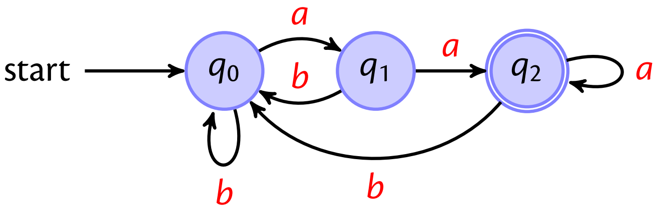
A language is regular iff there exists a regular expression that recognises all its strings.

A language is regular iff there exists an automaton that recognises all its strings.

for example $a^n b^n$ is not regular

Negation

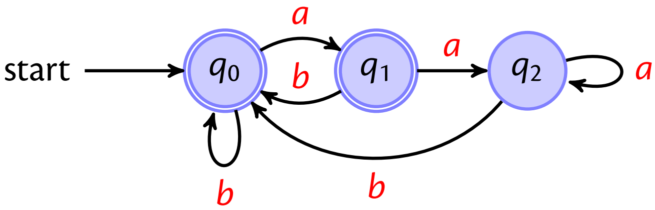
Regular languages are closed under negation:



But requires that the automaton is completed!

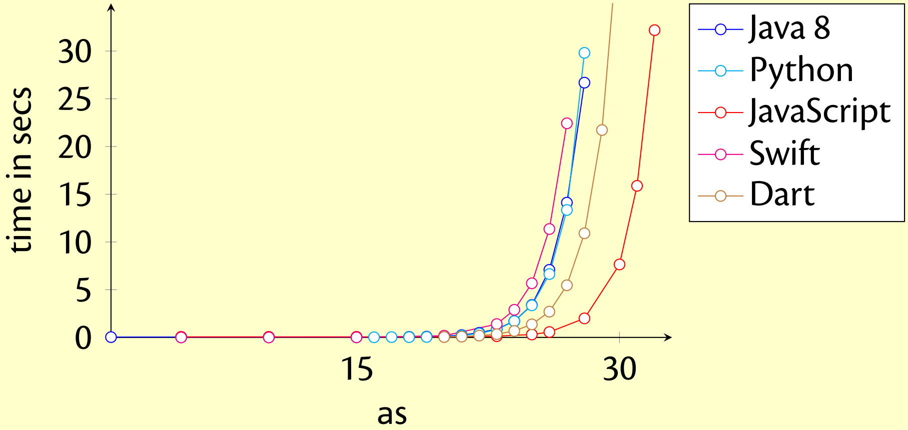
Negation

Regular languages are closed under negation:

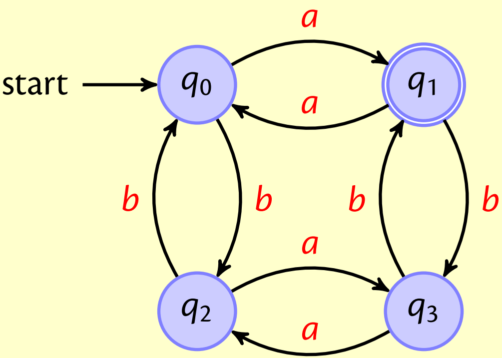


But requires that the automaton is completed!

$(a^*)^* \cdot b$



time in secs
0
5
10
15
20
25
30
15
30
as
Java 8
Python
JavaScript
Swift
Dart



Which language?

CW1: Regexes and L-function

Given

$$
\begin{array}{lll}
r^+ & L(r^+) & \stackrel{\text{def}}{=} \bigcup_{1\leq i}.\,L(r)^i \\
r^? & L(r^?) & \stackrel{\text{def}}{=} L(r) \cup \{[]\} \\
r_1 \,\&\, r_2 & L(r_1 \& r_2) & \stackrel{\text{def}}{=} L(r_1) \cap L(r_2) \\
r^{\{n\}} & L(r^{\{n\}}) & \stackrel{\text{def}}{=} L(r)^n \\
r^{\{..m\}} & L(r^{\{..m\}}) & \stackrel{\text{def}}{=} \bigcup_{0\leq i\leq m}.\,L(r)^i \\
r^{\{n..\}} & L(r^{\{n..\}}) & \stackrel{\text{def}}{=} \bigcup_{n\leq i}.\,L(r)^i \\
r^{\{n..m\}} & L(r^{\{n..m\}}) & \stackrel{\text{def}}{=} \bigcup_{n\leq i\leq m}.\,L(r)^i \\
\sim r & L(\sim r) & \stackrel{\text{def}}{=} \Sigma^* - L(r)
\end{array}
$$

Nullable

$$
\begin{array}{lcl}
nullable(r^{+}) & \stackrel{\text{def}}{=} & nullable(r) \\
nullable(r^{?}) & \stackrel{\text{def}}{=} & true \\
nullable(r_1 \,\&\, r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \wedge nullable(r_2) \\
nullable(r^{\{n\}}) & \stackrel{\text{def}}{=} & if\ n = 0\ then\ true\ else\ nullable(r) \\
nullable(r^{\{..m\}}) & \stackrel{\text{def}}{=} & true \\
nullable(r^{\{n..\}}) & \stackrel{\text{def}}{=} & if\ n = 0\ then\ true\ else\ nullable(r) \\
nullable(r^{\{n..m\}}) & \stackrel{\text{def}}{=} & if\ n = 0\ then\ true\ else\ nullable(r) \\
nullable(\sim r) & \stackrel{\text{def}}{=} & !\,nullable(r)
\end{array}
$$

Derivative

$$
\begin{array}{lcl}
der\,c\,(r^{+}) & \stackrel{\text{def}}{=} & (der\,c\,r) \cdot r^{*} \\
der\,c\,(r^{?}) & \stackrel{\text{def}}{=} & der\,c\,r \\
der\,c\,(r_1 \,\&\, r_2) & \stackrel{\text{def}}{=} & (der\,c\,r_1) \,\&\, (der\,c\,r_2) \\
der\,c\,(r^{\{n\}}) & \stackrel{\text{def}}{=} & if\ n = 0\ then\ \mathbf{0}\ else\ (der\,c\,r) \cdot r^{\{n-1\}} \\
der\,c\,(r^{\{..m\}}) & \stackrel{\text{def}}{=} & if\ m = 0\ then\ \mathbf{0}\ else\ (der\,c\,r) \cdot r^{\{..m-1\}} \\
der\,c\,(r^{\{n..\}}) & \stackrel{\text{def}}{=} & if\ n = 0\ then\ (der\,c\,r) \cdot r^{*}\ else\ (der\,c\,r) \cdot r^{\{n-1..\}} \\
der\,c\,(r^{\{n..m\}}) & \stackrel{\text{def}}{=} & if\ n = 0 \wedge m = 0\ then\ \mathbf{0}\ else \\
 & & if\ n = 0\ then\ (der\,c\,r) \cdot r^{\{..m-1\}}\ else\ (der\,c\,r) \cdot r^{\{n-1..m-1\}} \\
der\,c\,(\sim r) & \stackrel{\text{def}}{=} & \sim (der\,c\,r)
\end{array}
$$