

Compilers and Formal Languages

Email: christian.urban at kcl.ac.uk

Office Hour: Fridays 11 – 12

Location: N7.07 (North Wing, Bush House)

Slides & Progs: KEATS

Pollev: <https://pollev.com/cfltutoratki576>

1 Introduction, Languages	6 While-Language
2 Regular Expressions, Derivatives	7 Compilation, JVM
3 Automata, Regular Languages	8 Compiling Functional Languages
4 Lexing, Tokenising	9 Optimisations
5 Grammars, Parsing	10 LLVM

(Basic) Regular Expressions

$r ::=$	0	nothing
	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

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 !

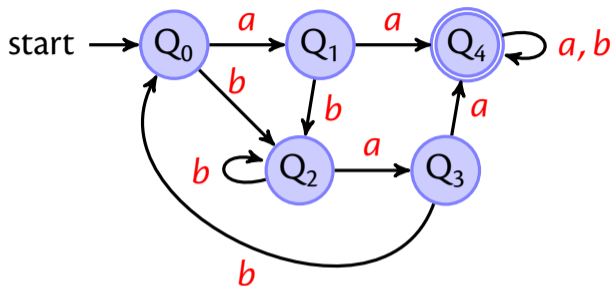
Automata

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

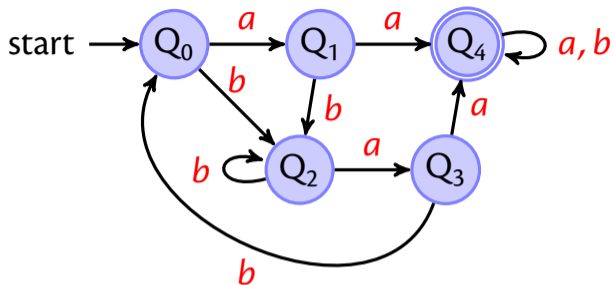
- an alphabet Σ
- a set of states Q_s
- one of these states is the start state Q_0
- some states are accepting states F , and
- there is transition function δ

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

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



- 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

$$\begin{array}{lll}
 (Q_0, a) \rightarrow Q_1 & (Q_1, a) \rightarrow Q_4 & (Q_4, a) \rightarrow Q_4 \\
 (Q_0, b) \rightarrow Q_2 & (Q_1, b) \rightarrow Q_2 & (Q_4, b) \rightarrow Q_4 \quad \dots
 \end{array}$$

Accepting a String

Given

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

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

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

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

$$N(\Sigma, Q_s, Q_{s_0}, F, \rho)$$

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

- a finite set of states, Q_s
- some these states are the start states, Q_{s_0}
- some states are accepting states, and
- there is transition **relation**, ρ

$$\begin{aligned}(Q_1, a) &\rightarrow Q_2 \\ (Q_1, a) &\rightarrow Q_3 \quad \dots\end{aligned}$$

Non-Deterministic Finite Automata

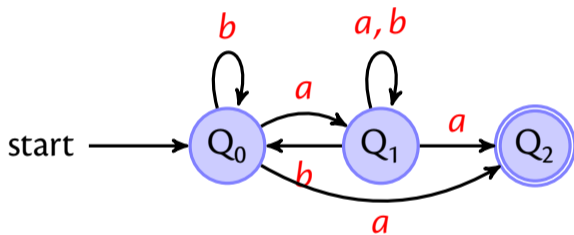
$N(\Sigma, Q_s, Q_{s_0}, F, \rho)$

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

- a finite set of states, Q_s
- some these states are the start states, Q_{s_0}
- some states are accepting states, and
- there is transition **relation**, ρ

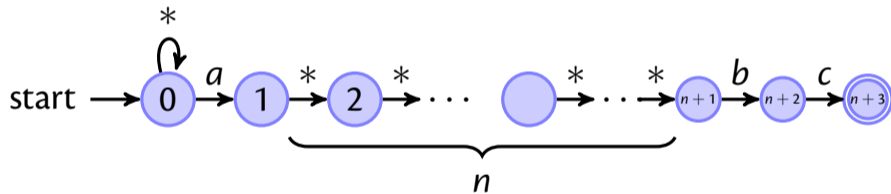
$$\begin{array}{l} (Q_1, a) \rightarrow Q_2 \\ (Q_1, a) \rightarrow Q_3 \end{array} \dots (Q_1, a) \rightarrow \{Q_2, Q_3\}$$

An NFA Example



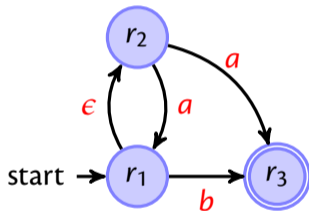
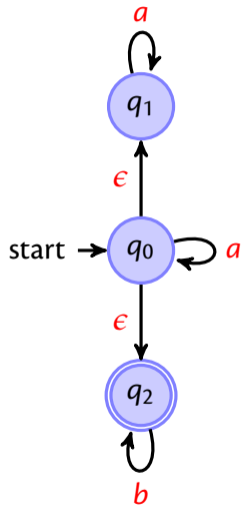
Another Example

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



Note the star-transitions: accept any character.

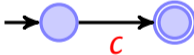
Two Epsilon NFA Examples



Thompson: Rexp to ϵ NFA

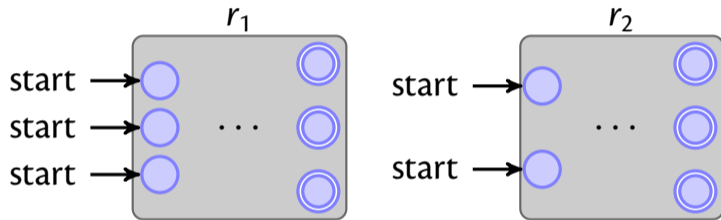
0 start \rightarrow 

1 start \rightarrow 

c start \rightarrow 

Case $r_1 \cdot r_2$

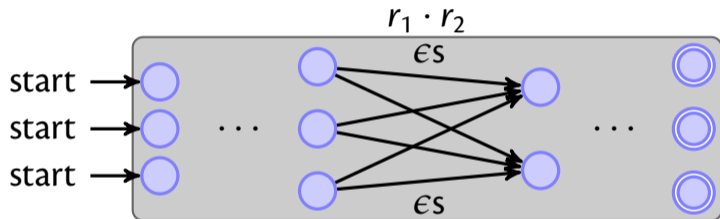
By recursion we are given two automata:



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 $r_1 \cdot r_2$

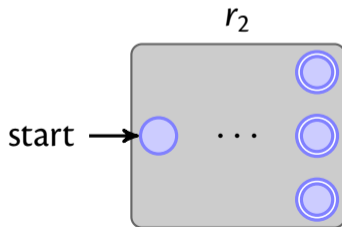
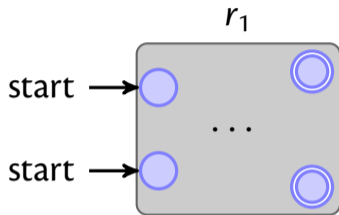
By recursion we are given two automata:



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 $r_1 + r_2$

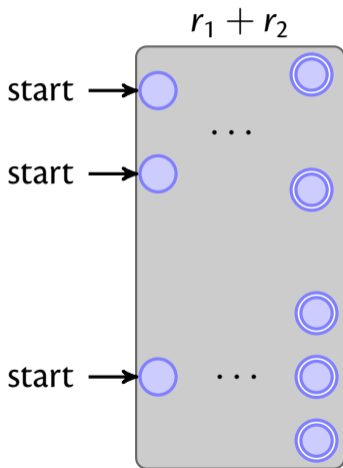
By recursion we are given two automata:



We can just put both automata together.

Case $r_1 + r_2$

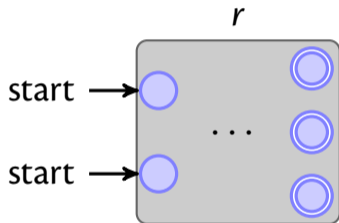
By recursion we are given two automata:



We can just put both automata together.

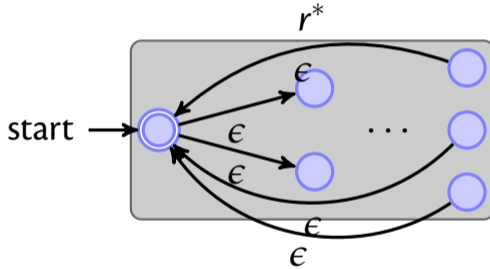
Case r^*

By recursion we are given an automaton for r :



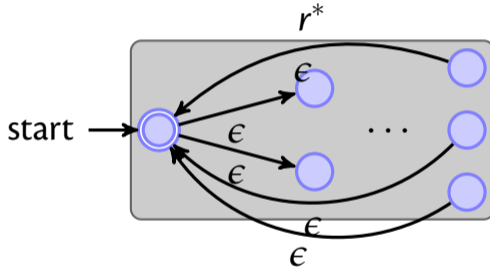
Case r^*

By recursion we are given an automaton for r :



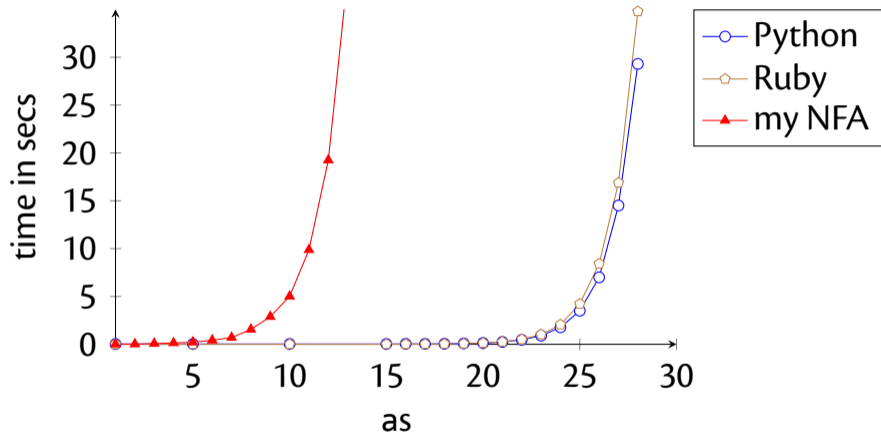
Case r^*

By recursion we are given an automaton for r :

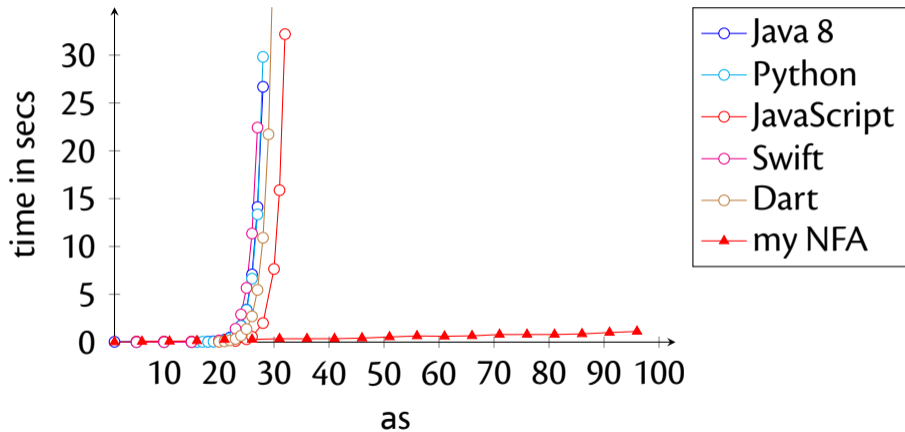


Why can't we just have an epsilon transition from the accepting states to the starting state?

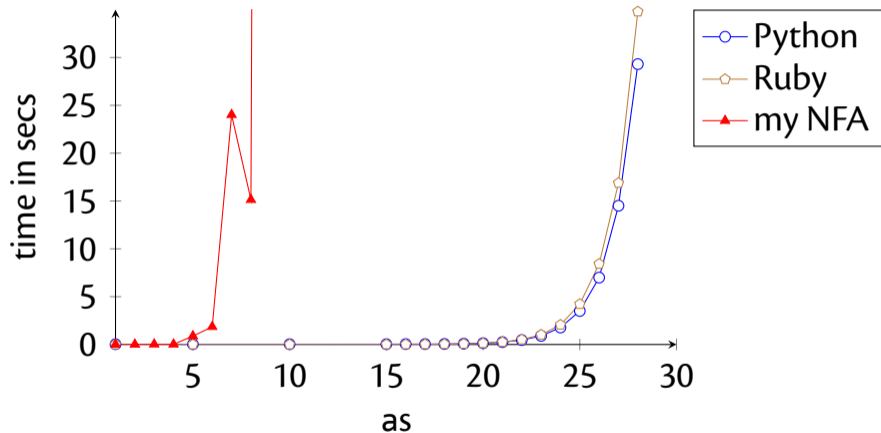
NFA Breadth-First: $a^?{n} \cdot a{n}$



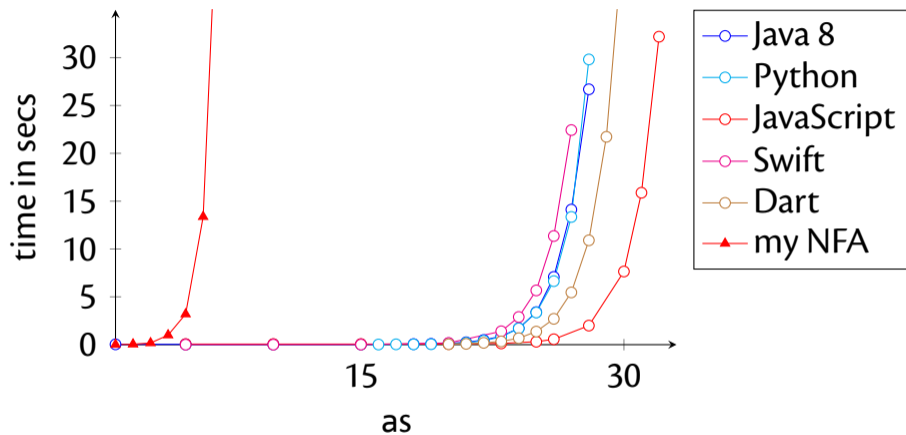
NFA Breadth-First: $(a^*)^* \cdot b$



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

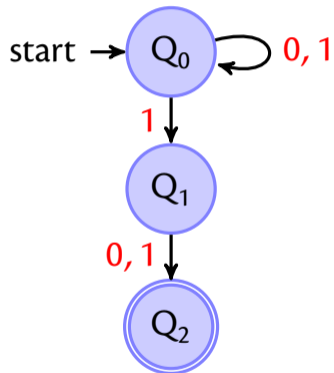


NFA Depth-First: $(a^*)^* \cdot b$



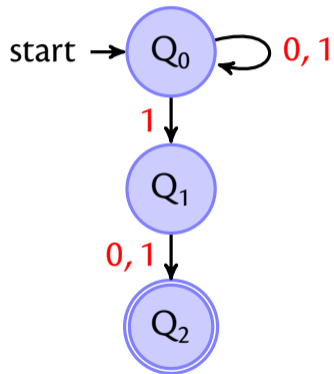
The punchline is that many existing libraries do depth-first search in NFAs (with backtracking).

Subset Construction



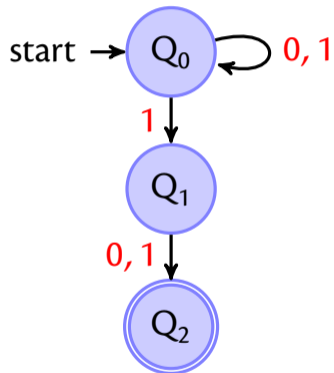
nodes	0	1
$\{\}$		
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

Subset Construction



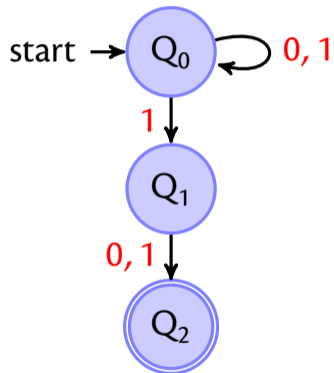
nodes	0	1
$\{\}$	$\{\}$	$\{\}$
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

Subset Construction



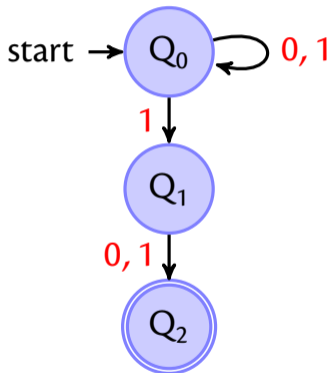
nodes	0	1
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0\}$	$\{0, 1\}$
$\{1\}$	$\{2\}$	$\{2\}$
$\{2\}$	$\{\}$	$\{\}$
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

Subset Construction



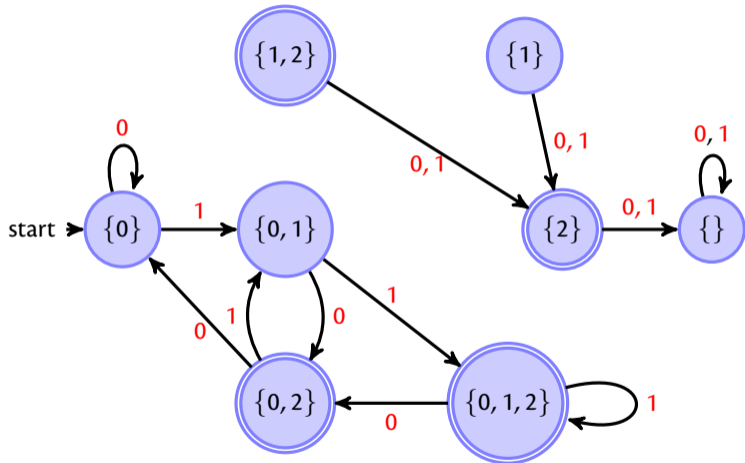
nodes	0	1
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0\}$	$\{0, 1\}$
$\{1\}$	$\{2\}$	$\{2\}$
$\{2\}$	$\{\}$	$\{\}$
$\{0, 1\}$	$\{0, 2\}$	$\{0, 1, 2\}$
$\{0, 2\}$	$\{0\}$	$\{0, 1\}$
$\{1, 2\}$	$\{2\}$	$\{2\}$
$\{0, 1, 2\}$	$\{0, 2\}$	$\{0, 1, 2\}$

Subset Construction



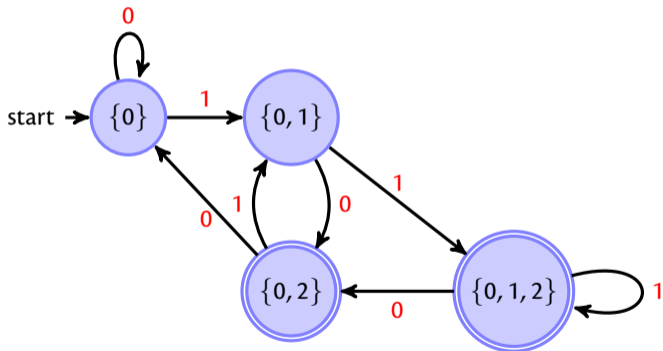
nodes	0	1
$\{\}$	$\{\}$	$\{\}$
s: $\{0\}$	$\{0\}$	$\{0, 1\}$
$\{1\}$	$\{2\}$	$\{2\}$
$\{2\}^*$	$\{\}$	$\{\}$
$\{0, 1\}$	$\{0, 2\}$	$\{0, 1, 2\}$
$\{0, 2\}^*$	$\{0\}$	$\{0, 1\}$
$\{1, 2\}^*$	$\{2\}$	$\{2\}$
$\{0, 1, 2\}^*$	$\{0, 2\}$	$\{0, 1, 2\}$

The Result

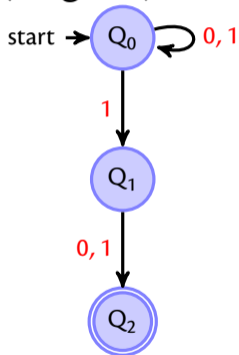


Removing Dead States

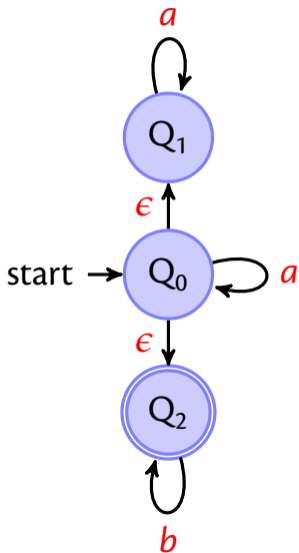
DFA:



(original) NFA:

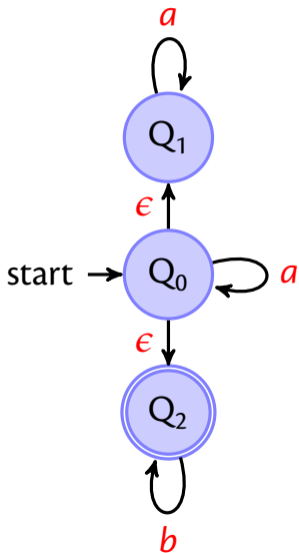


Subset Construction (ϵ NFA)



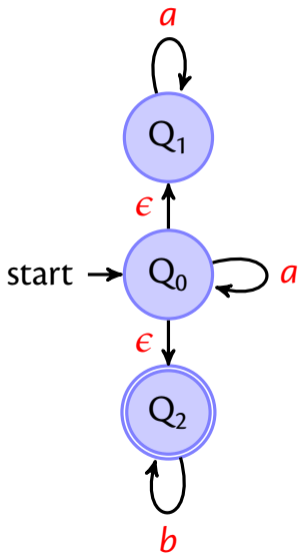
nodes	a	b
$\{\}$		
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

Subset Construction (ϵ NFA)



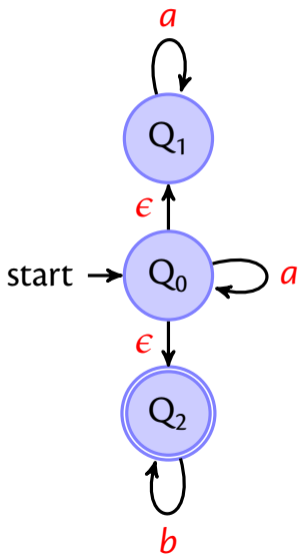
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

Subset Construction (ϵ NFA)



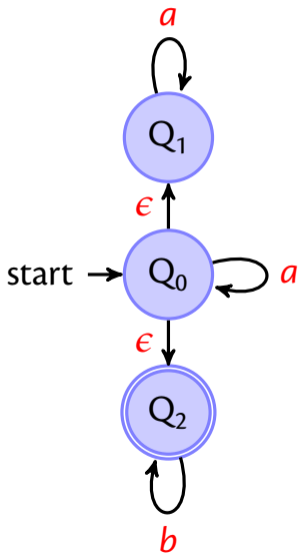
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

Subset Construction (ϵ NFA)



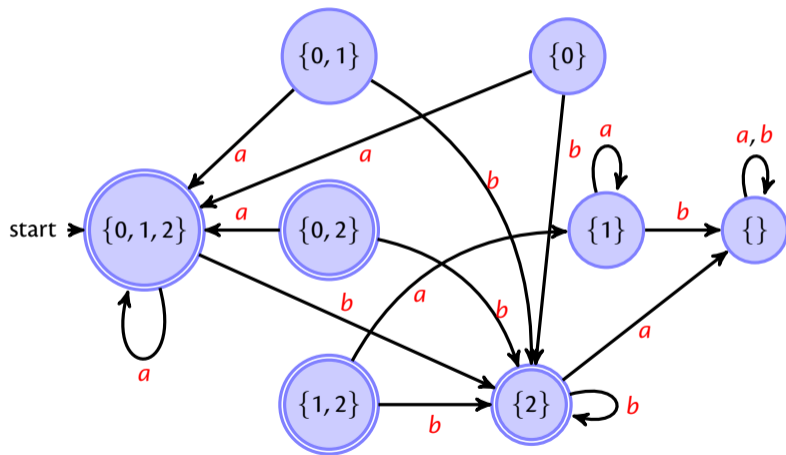
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0, 1\}$	$\{0, 1, 2\}$	$\{2\}$
$\{0, 2\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1, 2\}$	$\{1\}$	$\{2\}$
$\{0, 1, 2\}$	$\{0, 1, 2\}$	$\{2\}$

Subset Construction (ϵ NFA)



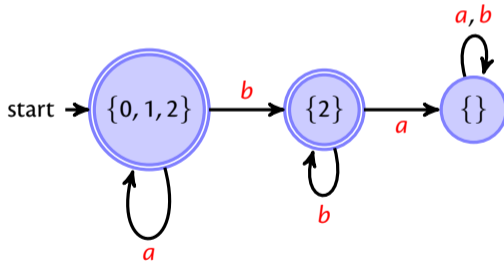
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}^*$	$\{\}$	$\{2\}$
$\{0, 1\}$	$\{0, 1, 2\}$	$\{2\}$
$\{0, 2\}^*$	$\{0, 1, 2\}$	$\{2\}$
$\{1, 2\}^*$	$\{1\}$	$\{2\}$
s: $\{0, 1, 2\}^*$	$\{0, 1, 2\}$	$\{2\}$

The Result

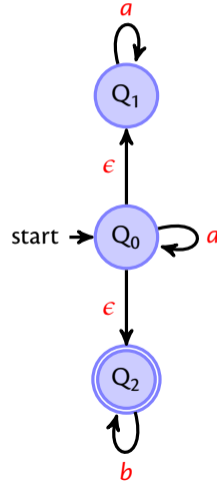


Removing Dead States

DFA:



(original) NFA:

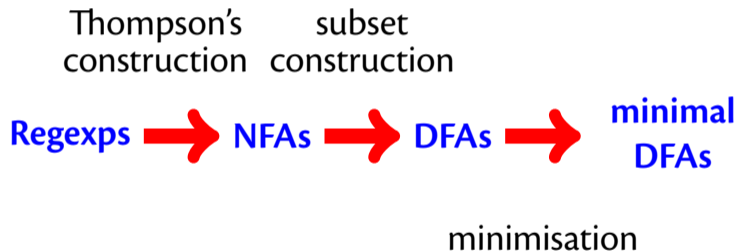


Regexps and Automata

Thompson's construction subset construction

Regexps  NFAs  DFAs

Regexps and Automata



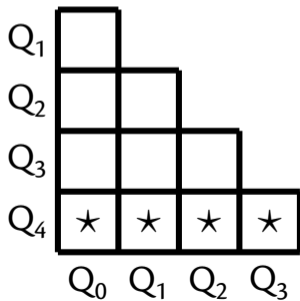
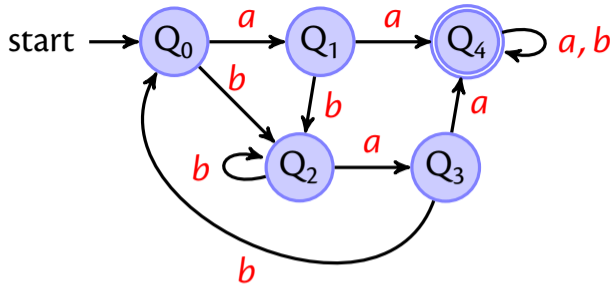
DFA Minimisation

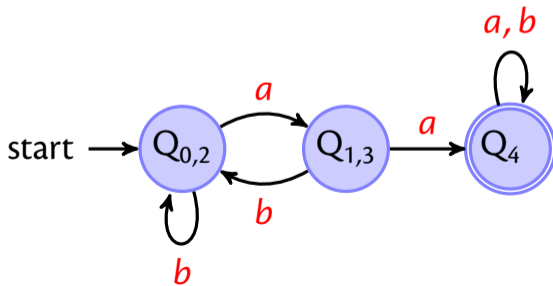
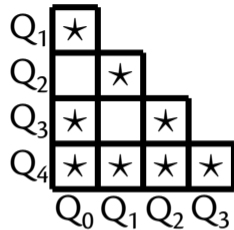
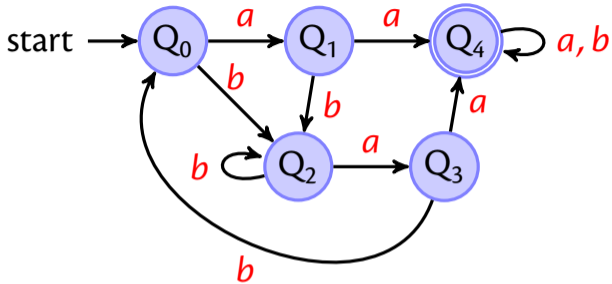
1. Take all pairs (q, p) with $q \neq p$
2. Mark all pairs that accepting and non-accepting states
3. For all unmarked pairs (q, p) and all characters c test whether

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

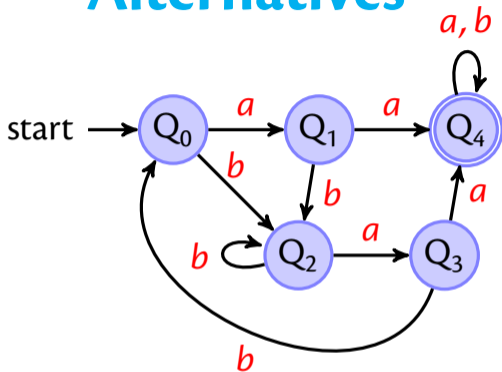
are marked. If yes in at least one case, then also mark (q, p) .

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



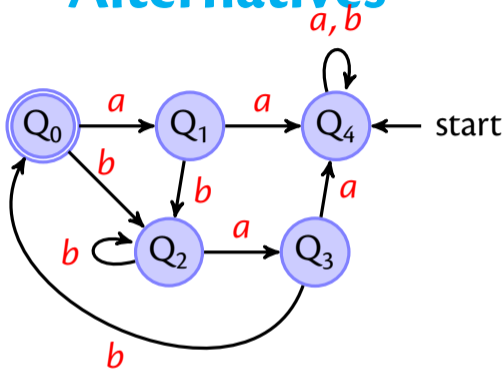


Alternatives



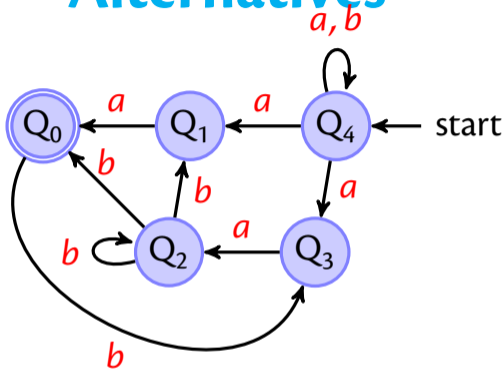
- exchange initial / accepting states

Alternatives



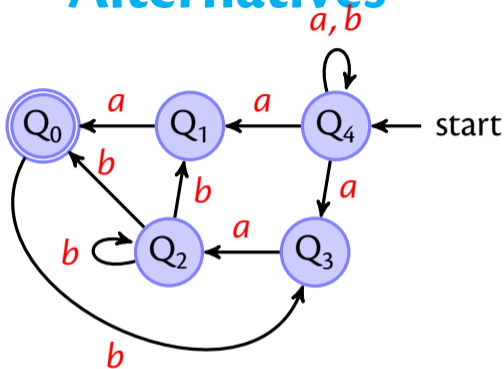
- exchange initial / accepting states
- reverse all edges

Alternatives



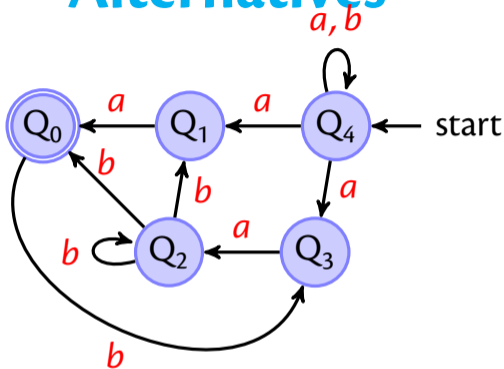
- exchange initial / accepting states
- reverse all edges
- subset construction \Rightarrow DFA

Alternatives



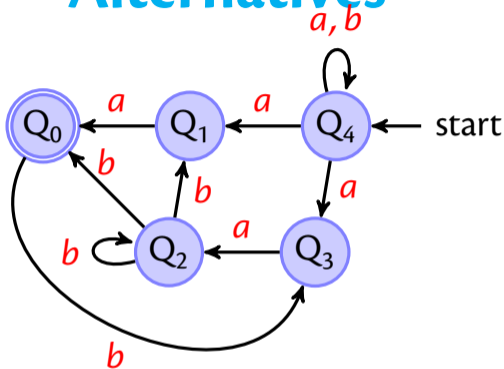
- exchange initial / accepting states
- reverse all edges
- subset construction \Rightarrow DFA
- remove dead states

Alternatives



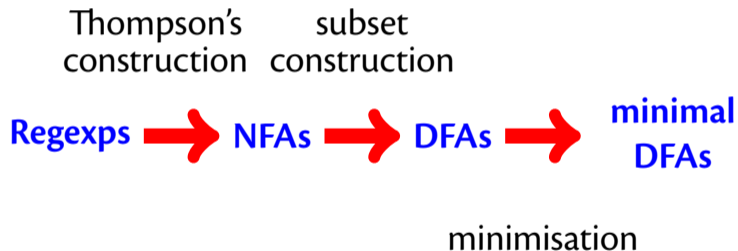
- exchange initial / accepting states
- reverse all edges
- subset construction \Rightarrow DFA
- remove dead states
- repeat once more

Alternatives

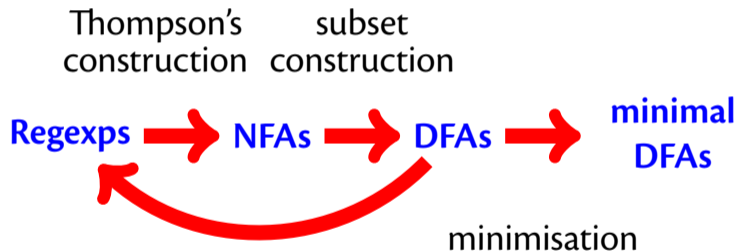


- exchange initial / accepting states
- reverse all edges
- subset construction \Rightarrow DFA
- remove dead states
- repeat once more \Rightarrow minimal DFA

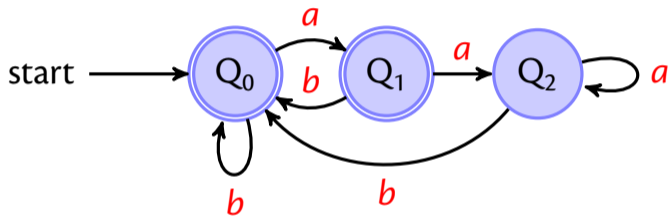
Regexps and Automata

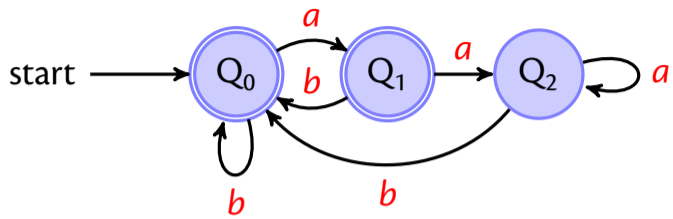


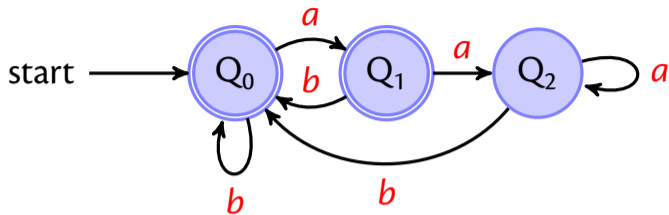
Regexps and Automata



DFA to Rexp





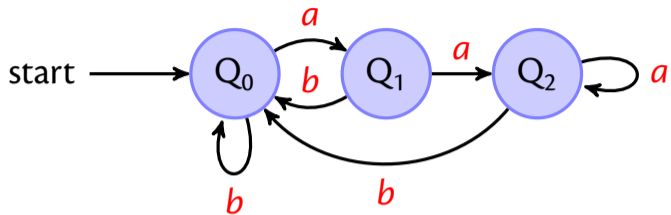


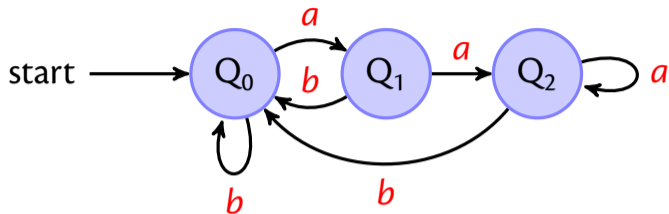
You know how to solve since school days, no?

$$Q_0 = 2Q_0 + 3Q_1 + 4Q_2$$

$$Q_1 = 2Q_0 + 3Q_1 + 1Q_2$$

$$Q_2 = 1Q_0 + 5Q_1 + 2Q_2$$





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

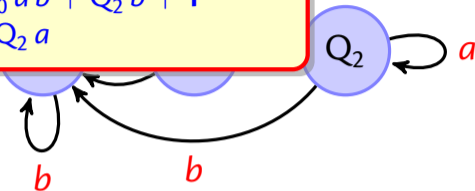
$$Q_1 = Q_0 a$$

$$Q_2 = Q_1 a + Q_2 a$$

substitute Q_1 into Q_0 & Q_2 :

$$Q_0 = Q_0 b + Q_0 a b + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$



$$Q_0 = Q_0 b + Q_1 b + Q_2 b + 1$$

$$Q_1 = Q_0 a$$

$$Q_2 = Q_1 a + Q_2 a$$

substitute Q_1 into Q_0 & Q_2 :

$$Q_0 = Q_0 b + Q_0 a b + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$



simplifying Q_0 :

$$Q_0 = Q_0 (b + a b) + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$

$$Q_0 = Q_0 b + Q_1 b + Q_2 b + 1$$

$$Q_1 = Q_0 a$$

$$Q_2 = Q_1 a + Q_2 a$$

substitute Q_1 into Q_0 & Q_2 :

$$Q_0 = Q_0 b + Q_0 a b + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$



simplifying Q_0 :

$$Q_0 = Q_0 (b + a b) + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$

$$Q_0 = Q_0 b + Q_1 b + Q_2 b + 1$$

$$Q_1 = Q_0 a$$

Arden's Lemma:

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

substitute Q_1 into Q_0 & Q_2 :

$$Q_0 = Q_0 b + Q_0 a b + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$



simplifying Q_0 :

$$Q_0 = Q_0 (b + a b) + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$

Arden for Q_2 :

$$Q_0 = Q_0 (b + a b) + Q_2 b + 1$$

$$Q_2 = Q_0 a a (a^*)$$

$$Q_1 = Q_0 a$$

Arden's Lemma:

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

substitute Q_1 into Q_0 & Q_2 :

$$Q_0 = Q_0 b + Q_0 a b + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$



simplifying Q_0 :

$$Q_0 = Q_0 (b + a b) + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$

Arden for Q_2 :

$$Q_0 = Q_0 (b + a b) + Q_2 b + 1$$

$$Q_2 = Q_0 a a (a^*)$$

$Q_1 =$
 $Q_2 =$

Substitute Q_2 and simplify:

$$Q_0 = Q_0 (b + a b + a a (a^*) b) + 1$$

substitute Q_1 into Q_0 & Q_2 :

$$Q_0 = Q_0 b + Q_0 a b + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$



simplifying Q_0 :

$$Q_0 = Q_0 (b + a b) + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$

Arden's Lemma:

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

1

$$\begin{aligned} Q_1 &= \\ Q_2 &= \end{aligned}$$

Substitute Q_2 and simplify:

$$Q_0 = Q_0 (b + a b + a a (a^*) b) + 1$$

Arden again for Q_0 :

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

substitute Q_1 into Q_0 & Q_2 :

$$Q_0 = Q_0 b + Q_0 a b + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$



simplifying Q_0 :

$$Q_0 = Q_0 (b + a b) + Q_2 b + 1$$

$$Q_2 = Q_0 a a + Q_2 a$$

Arden for Q_2 :

$$Q_0 = Q_0 (b + a b) + Q_2 b + 1$$

$$Q_2 = Q_0 a a (a^*)$$

$Q_1 =$
 $Q_2 =$

Substitute Q_2 into Q_0 & Q_1 :

$$Q_0 = Q_0$$

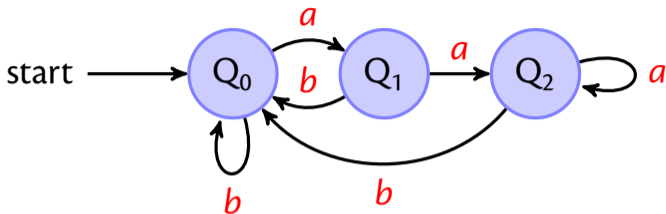
Finally:

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

$$Q_1 = (b + a b + a a (a^*) b)^* a$$

$$Q_2 = (b + a b + a a (a^*) b)^* a a (a^*)$$

Arden
 Q_0



$$Q_0 = Q_0 b + Q_1 b + Q_2 b + 1$$

$$Q_1 = Q_0 a$$

$$Q_2 = Q_1 a + Q_2 a$$

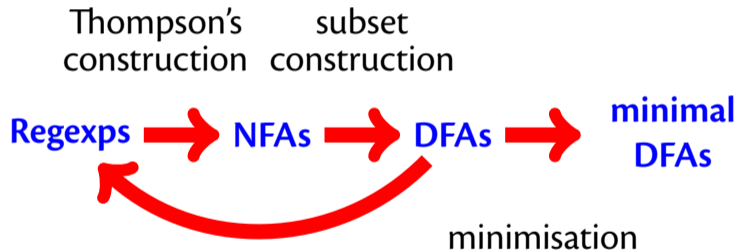
Finally:

$$Q_0 = (b + ab + aa(a^*)b)^*$$

$$Q_1 = (b + ab + aa(a^*)b)^* a$$

$$Q_2 = (b + ab + aa(a^*)b)^* aa(a^*)$$

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)

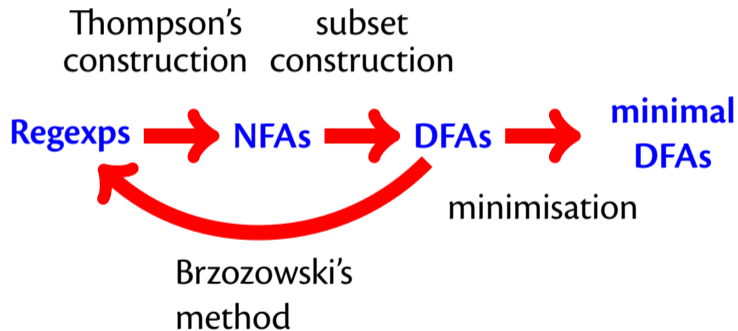
A language is **regular** iff there exists a regular expression that recognises all its strings.

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:

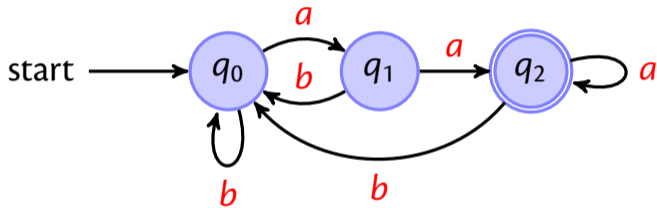
A language is **regular** iff there exists a regular expression that recognises all its strings.

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

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

Negation

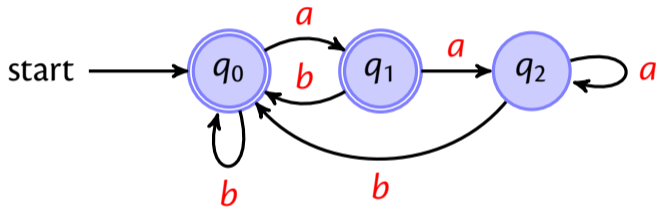
Regular languages are closed under negation:



But requires that the automaton is **completed!**

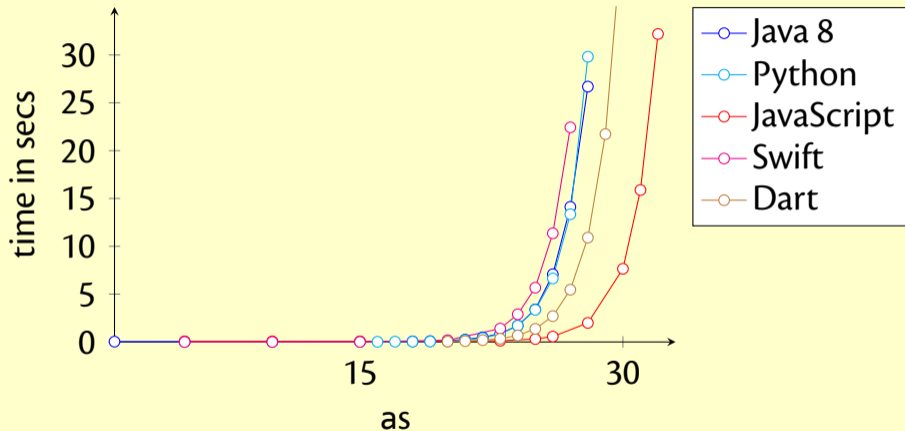
Negation

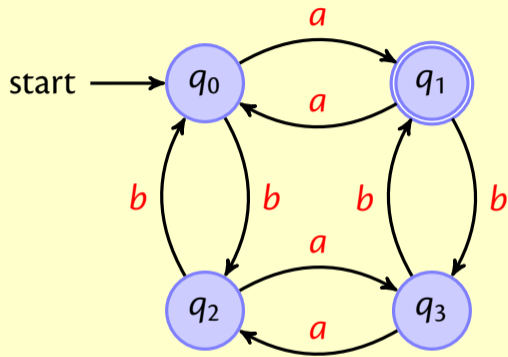
Regular languages are closed under negation:



But requires that the automaton is **completed!**

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





Which language?

I always thought dfa's needed a transition for each state for each character, and if not it would be an nfa not a dfa. Is there an example that disproves this?

Do the regular expression matchers in Python and Java 8 have more features than the one implemented in this module? Or is there another reason for their inefficiency?

- CW
- power law / proof
- CW feedback
- too polished CW submissions