Compilers and Formal Languages

Email: christian.urban at kcl.ac.uk
Slides & Progs: KEATS (also homework is there)

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

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

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!

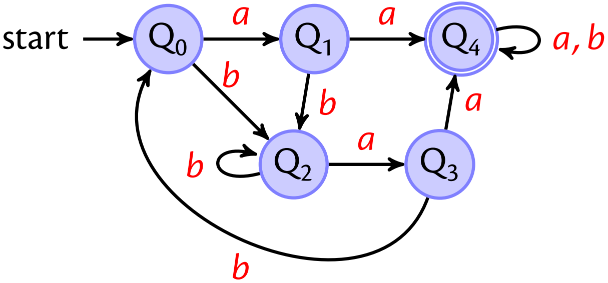
Automata

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

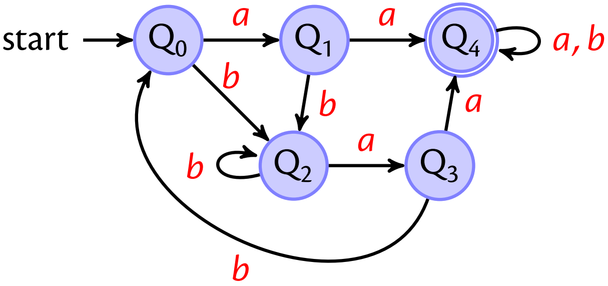
$$A(\Sigma, \mathrm{Qs}, \mathrm{Q}_0, F, \delta)$$



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

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

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

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

you can define

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

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

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

Non-Deterministic Finite Automata

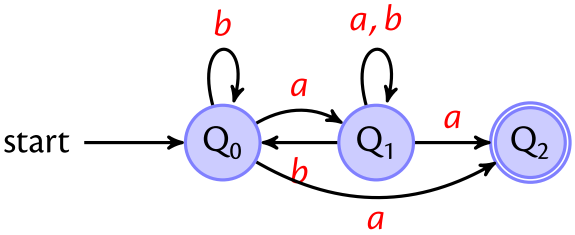
$N(\Sigma, Qs, Qs_0, F, \rho)$

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

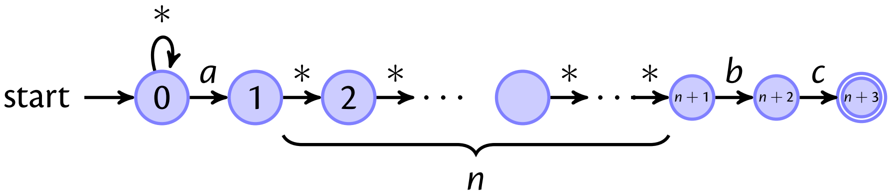
$$\begin{array}{l}(Q_1, a) \rightarrow Q_2 \\ (Q_1, a) \rightarrow Q_3\end{array} \quad \ldots \qquad (Q_1, a) \rightarrow \{Q_2, Q_3\}$$

An NFA Example



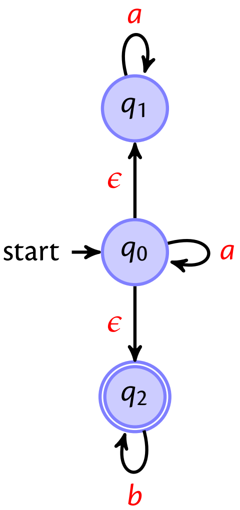
Another Example

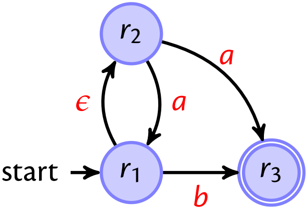
For the regular expression $(.^*)a\,(.^{\{n\}})bc$



Note the star-transitions: accept any character.

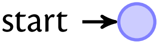
Two Epsilon NFA Examples



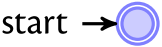


Thompson: Rexp to ϵNFA

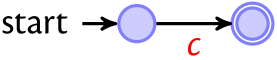
0



1

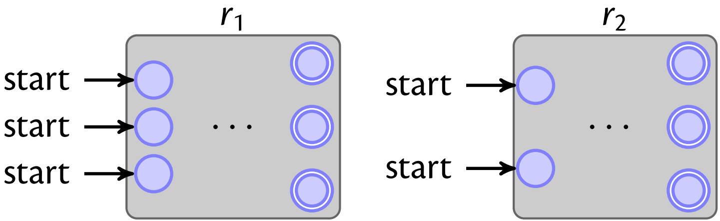


c



Case $r_1 \cdot r_2$

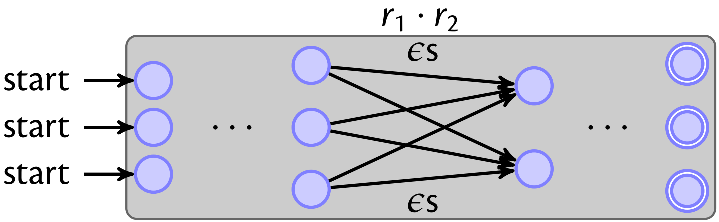
By recursion we are given two automata:



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 $r_1 \cdot r_2$

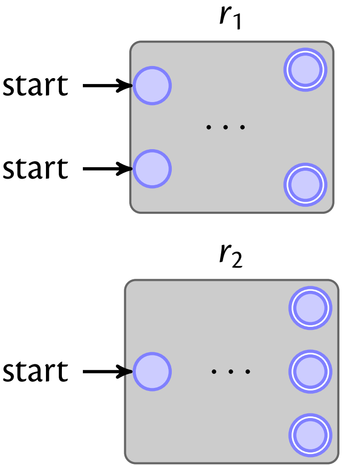
By recursion we are given two automata:



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 $r_1 + r_2$

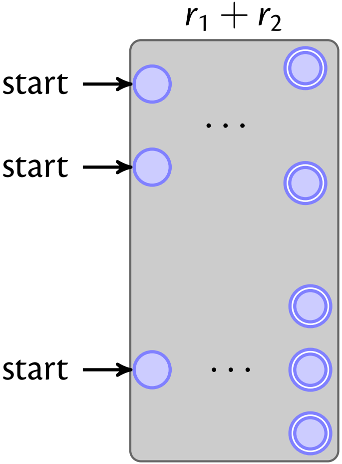
By recursion we are given two automata:



We can just put both automata together.

Case $r_1 + r_2$

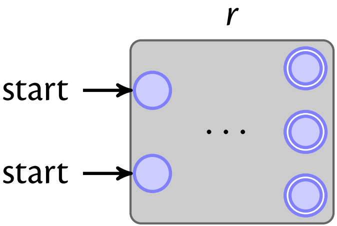
By recursion we are given two automata:



We can just put both automata together.

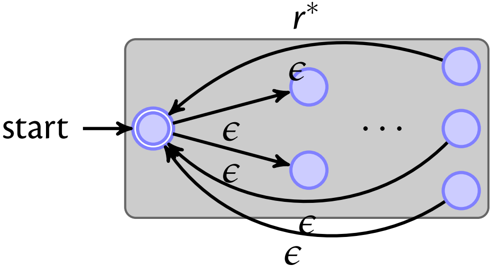
Case r^*

By recursion we are given an automaton for r:



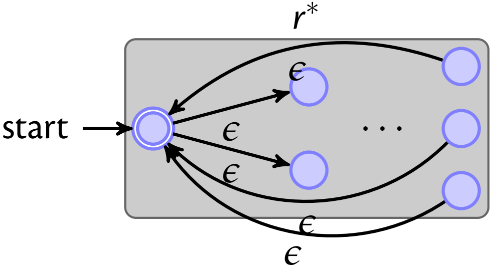
Case r^*

By recursion we are given an automaton for r:



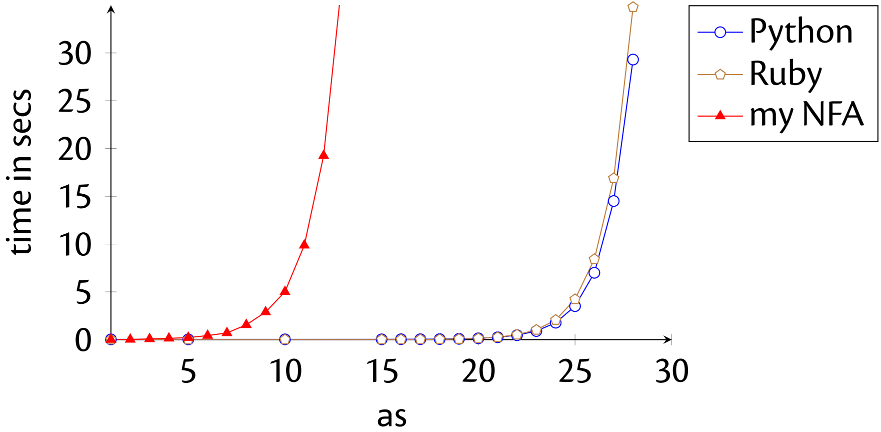
Case r^*

By recursion we are given an automaton for r:

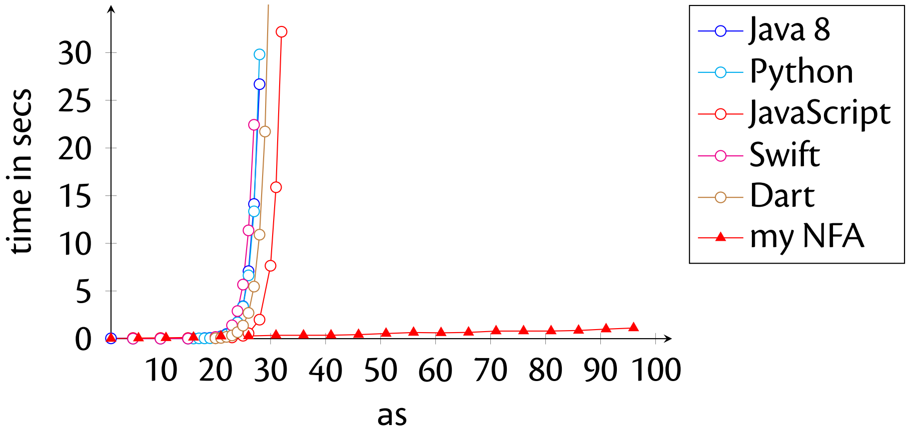


Why can't we just have an epsilon transition from the accepting states to the starting state?

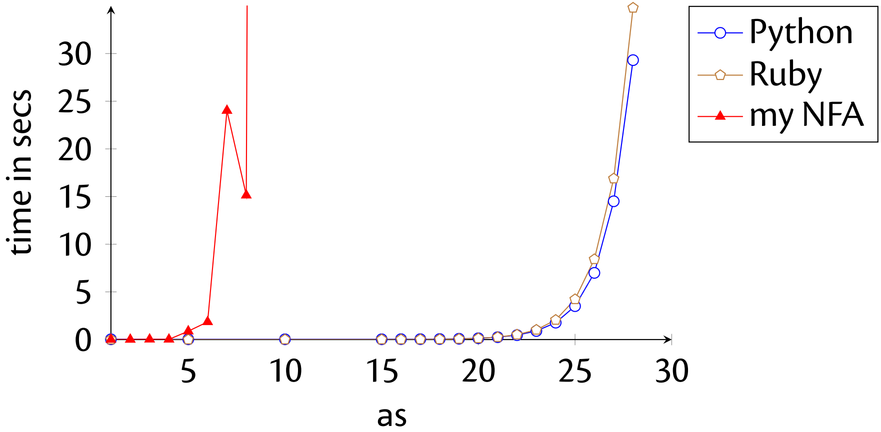
NFA Breadth-First: $a^{?\{n\}} \cdot a^{\{n\}}$



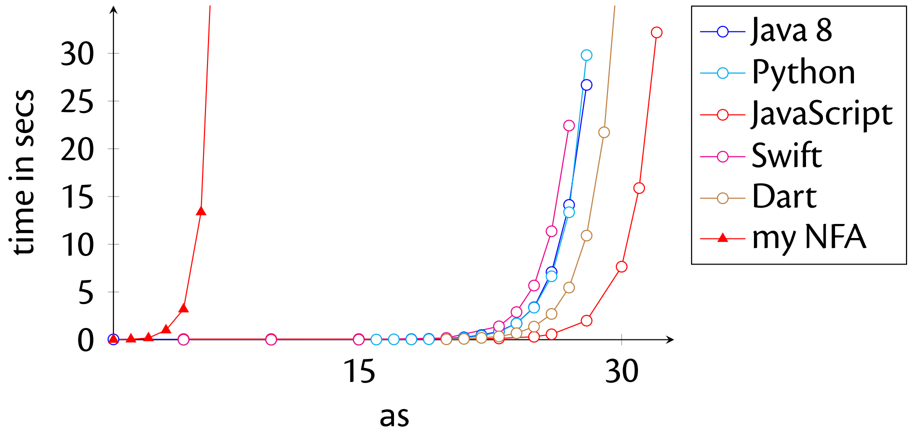
NFA Breadth-First: $(a^*)^* \cdot b$



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

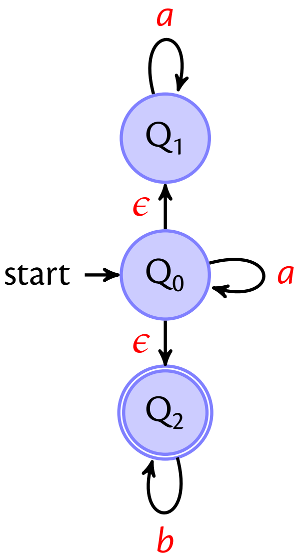


NFA Depth-First: $(a^*)^* \cdot b$



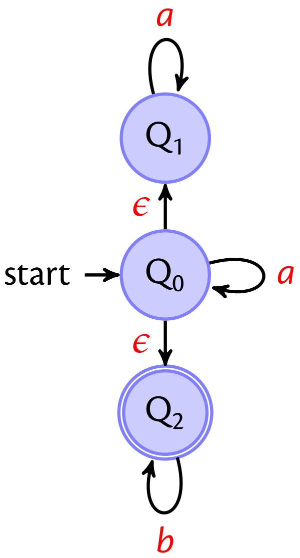
The punchline is that many existing libraries do depth-first search in NFAs (with backtracking).

Subset Construction



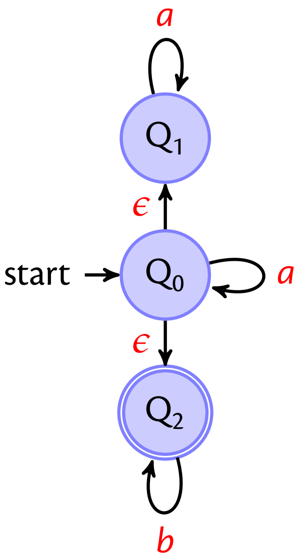
nodes	a	b
$\{\}$		
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

Subset Construction



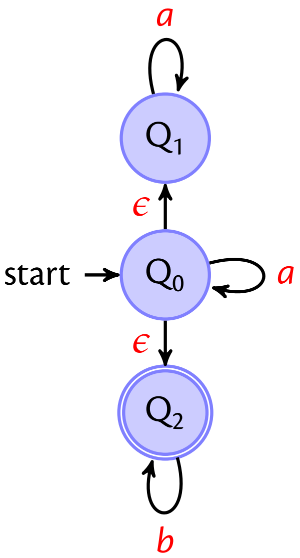
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



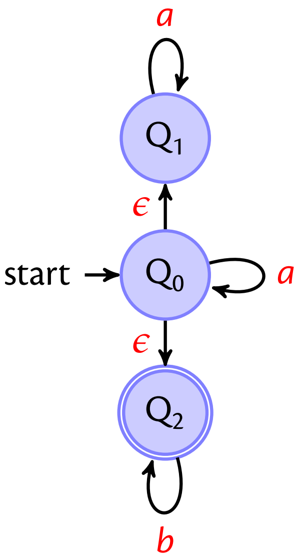
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



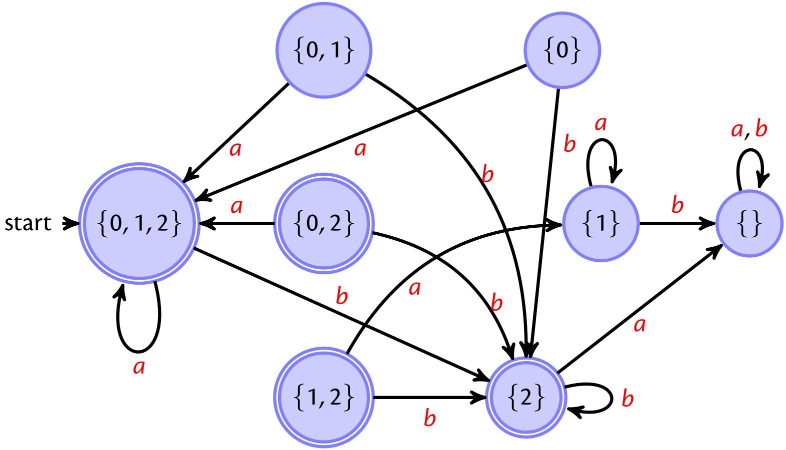
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$	$\{\}$	$\{2\}$
$\{0,1\}$	$\{0,1,2\}$	$\{2\}$
$\{0,2\}$	$\{0,1,2\}$	$\{2\}$
$\{1,2\}$	$\{1\}$	$\{2\}$
$\{0,1,2\}$	$\{0,1,2\}$	$\{2\}$

Subset Construction



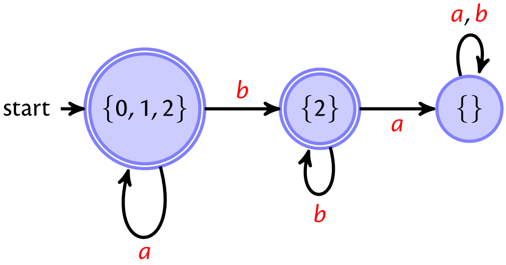
nodes	a	b
$\{\}$	$\{\}$	$\{\}$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\{\}$
$\{2\}$ *	$\{\}$	$\{2\}$
$\{0,1\}$	$\{0,1,2\}$	$\{2\}$
$\{0,2\}$ *	$\{0,1,2\}$	$\{2\}$
$\{1,2\}$ *	$\{1\}$	$\{2\}$
s: $\{0,1,2\}$ *	$\{0,1,2\}$	$\{2\}$

The Result

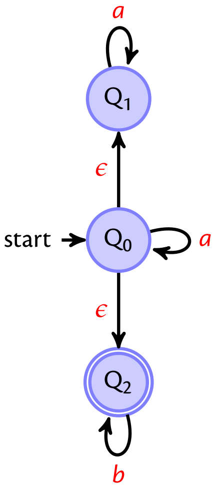


Removing Dead States

DFA:



(original) NFA:



Regexps and Automata

Thompson's construction | subset construction

Regexps → **NFAs** → **DFAs**

Regexps and Automata

Thompson's construction | subset construction

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

minimisation

DFA Minimisation

Take all pairs (q, p) with $q \neq p$

Mark all pairs that accepting and non-accepting states

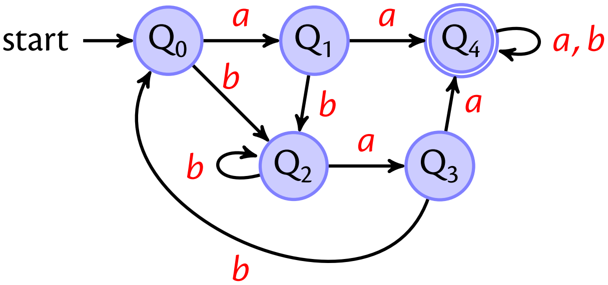
For all unmarked pairs (q, p) and all characters c test whether

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

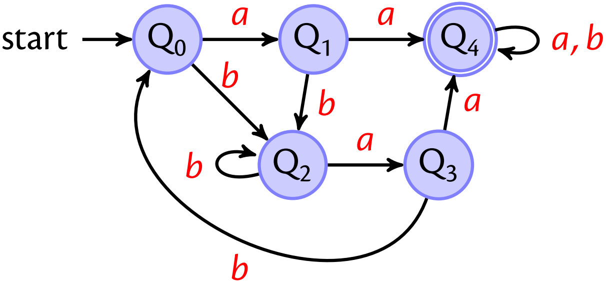
are marked. If yes in at least one case, then also mark (q, p).

Repeat last step until no change.

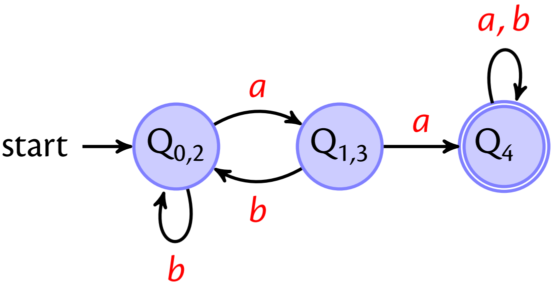
All unmarked pairs can be merged.



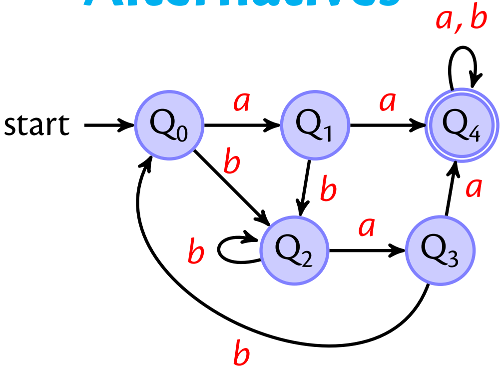
	Q_0	Q_1	Q_2	Q_3
Q_1				
Q_2				
Q_3				
Q_4	⋆	⋆	⋆	⋆



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

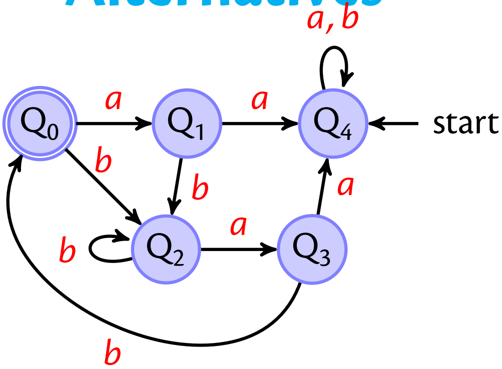


Alternatives



exchange initial / accepting states

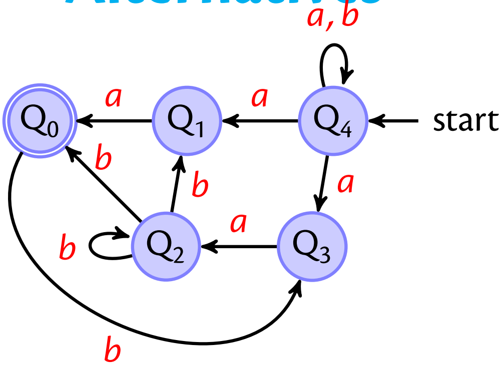
Alternatives



exchange initial / accepting states

reverse all edges

Alternatives

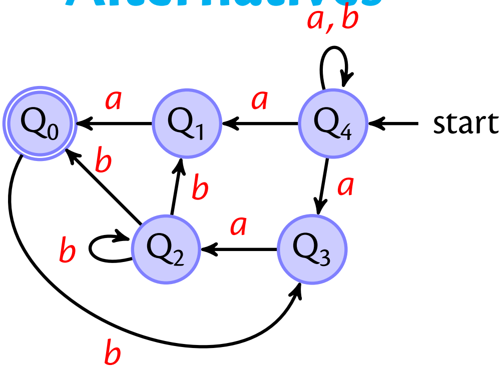


exchange initial / accepting states

reverse all edges

subset construction ⇒ DFA

Alternatives



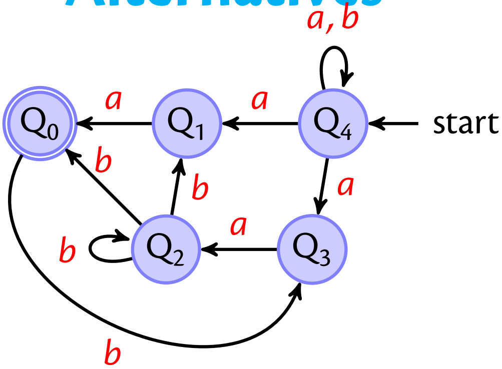
exchange initial / accepting states

reverse all edges

subset construction ⇒ DFA

remove dead states

Alternatives



exchange initial / accepting states

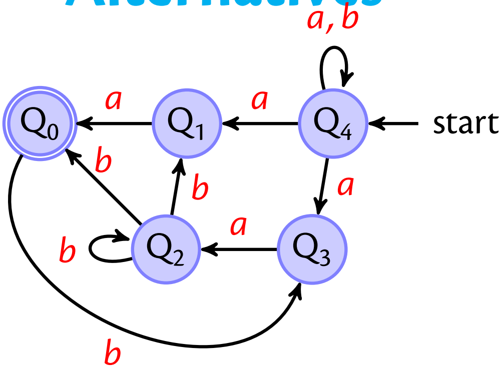
reverse all edges

subset construction $\Rightarrow$ DFA

remove dead states

repeat once more

Alternatives



exchange initial / accepting states

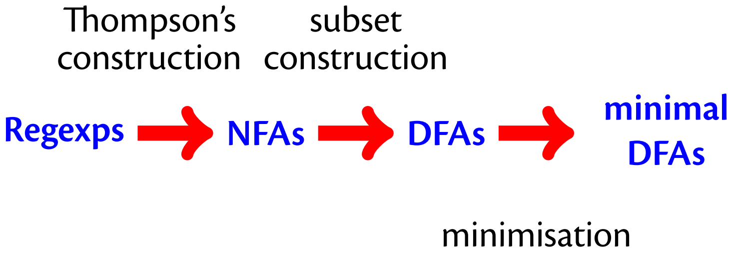
reverse all edges

subset construction ⇒ DFA

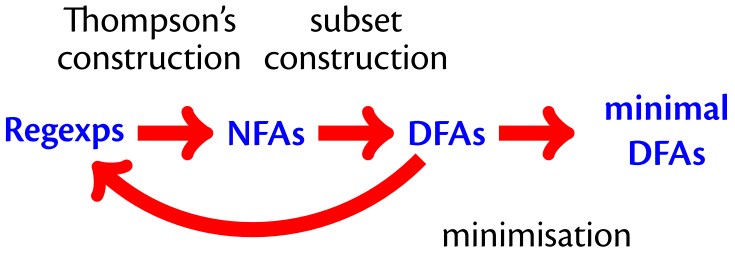
remove dead states

repeat once more ⇒ minimal DFA

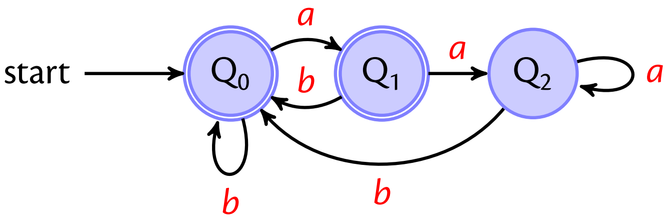
Regexps and Automata

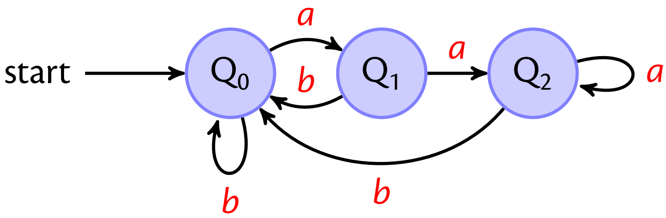


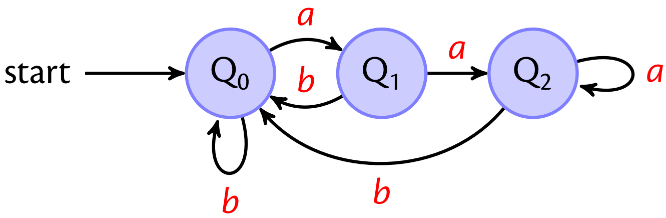
Regexps and Automata



DFA to Rexp

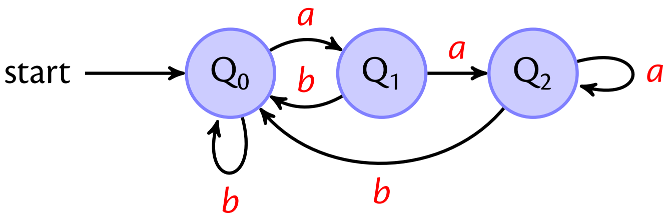


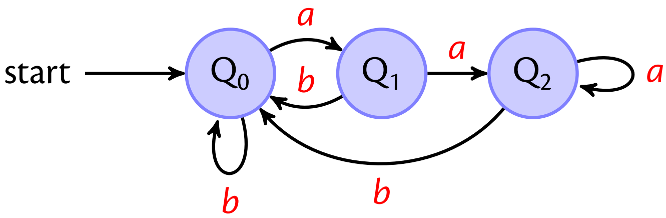

start
Q_0
Q_1
Q_2
a
a
a
b
b
b



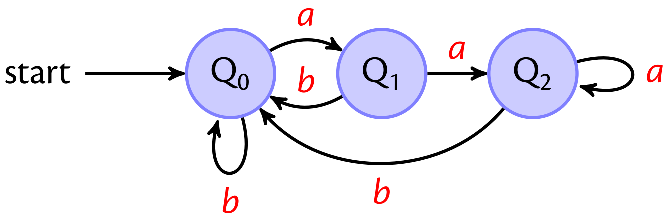
You know how to solve since school days, no?

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


start
Q_0
Q_1
Q_2
a
a
a
b
b
b



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

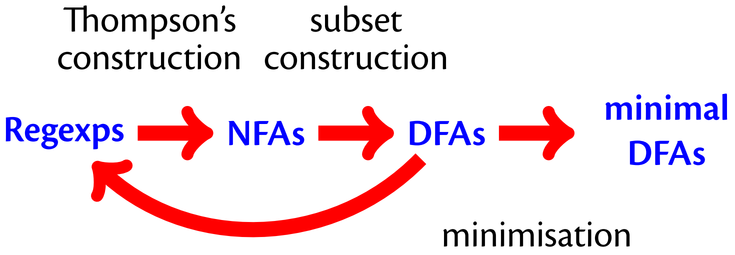


$$Q_0 = Q_0\, b + Q_1\, b + Q_2\, b + \mathbf{1}$$
$$Q_1 = Q_0\, a$$
$$Q_2 = Q_1\, a + Q_2\, a$$

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

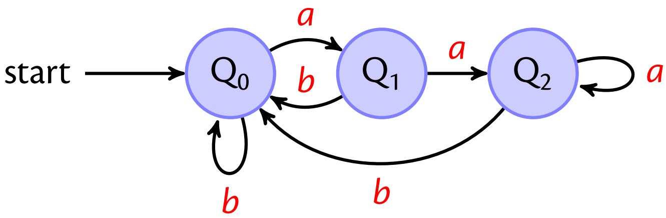
$$
\begin{aligned}
rev(\mathbf{0}) &\stackrel{\text{def}}{=} \mathbf{0}\\
rev(\mathbf{1}) &\stackrel{\text{def}}{=} \mathbf{1}\\
rev(c) &\stackrel{\text{def}}{=} c\\
rev(r_1 + r_2) &\stackrel{\text{def}}{=} rev(r_1) + rev(r_2)\\
rev(r_1 \cdot r_2) &\stackrel{\text{def}}{=} rev(r_2) \cdot rev(r_1)\\
rev(r^*)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

and the set

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

prove whether

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



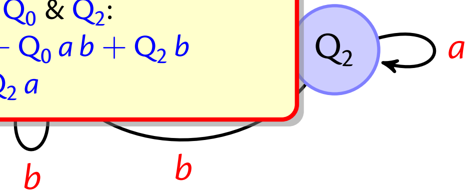
$$\begin{aligned}
Q_0 &= \mathbf{1} + Q_0\, b + Q_1\, b + Q_2\, b \\
Q_1 &= Q_0\, a \\
Q_2 &= Q_1\, a + Q_2\, a
\end{aligned}$$

Arden's Lemma:

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

substitute Q_1 into Q_0 & Q_2:

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



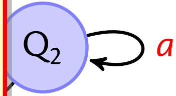
$$\begin{aligned} Q_0 &= \mathbf{1} + Q_0\, b + Q_1\, b + Q_2\, b \\ Q_1 &= Q_0\, a \\ Q_2 &= Q_1\, a + Q_2\, a \end{aligned}$$

Arden's Lemma:

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

substitute Q_1 into Q_0 & Q_2:

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



simplifying Q_0:

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

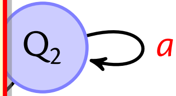
$$\begin{aligned} Q_0 &= \mathbf{1} + Q_0\,b + Q_1\,b + Q_2\,b \\ Q_1 &= Q_0\,a \\ Q_2 &= Q_1\,a + Q_2\,a \end{aligned}$$

Arden's Lemma:

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

substitute Q_1 into Q_0 & Q_2:

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



simplifying Q_0:

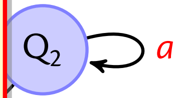
$$Q_0 = \mathbf{1} + Q_0\,(b + a\,b) + Q_2\,b$$
$$Q_2 = Q_0\,a\,a + Q_2\,a$$

Arden for Q_2:

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

Arden's Lemma:

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



substitute Q_1 into Q_0 & Q_2:

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

simplifying Q_0:

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

Arden for Q_2:

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

Substitute Q_2 and simplify:

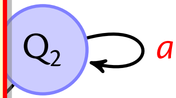
$$Q_0 = \mathbf{1} + Q_0\, (b + a\, b + a\, a\, (a^*)\, b)$$

Arden's Lemma:

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

substitute Q_1 into Q_0 & Q_2:

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



simplifying Q_0:

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

Arden for Q_2:

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

Substitute Q_2 and simplify:

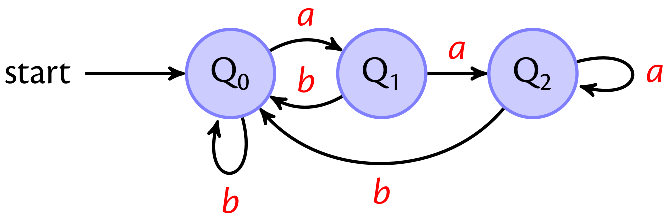
$$Q_0 = \mathbf{1} + Q_0\,(b + a\, b + a\, a\,(a^*)\, b)$$

Arden's Lemma:

If $q = q\, r$

Arden again for Q_0:

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



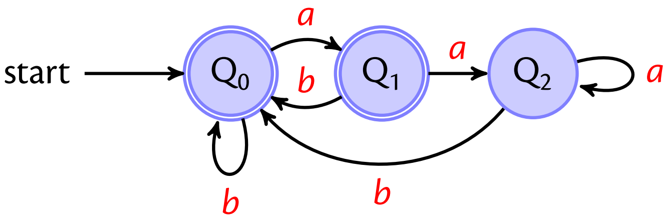
$$
\begin{aligned}
Q_0 &= \mathbf{1} + Q_0\, b + Q_1\, b + Q_2\, b \\
Q_1 &= Q_0\, a \\
Q_2 &= Q_1\, a + Q_2\, a
\end{aligned}
$$

Arden's Lemma:

If $q = q\, r + s$

Finally:

$$
\begin{aligned}
Q_0 &= (b + a\, b + a\, a\, (a^*)\, b)^* \\
Q_1 &= (b + a\, b + a\, a\, (a^*)\, b)^*\, a \\
Q_2 &= (b + a\, b + a\, a\, (a^*)\, b)^*\, a\, a\, (a^*)
\end{aligned}
$$



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

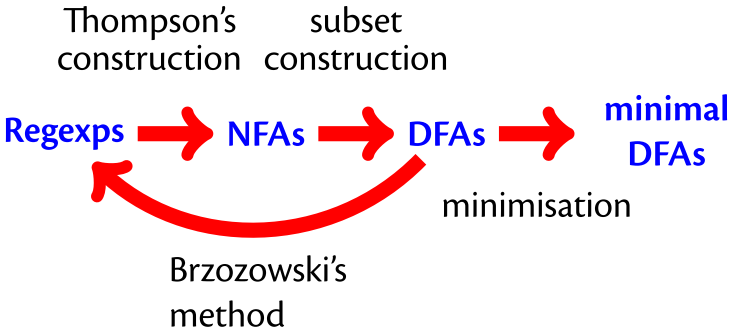
Arden's Lemma:

If $q = q\, r + s$

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

Two equivalent definitions:

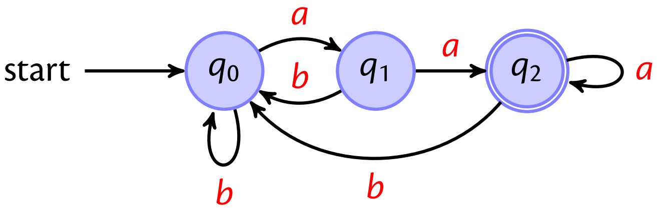
A language is regular iff there exists a regular expression that recognises all its strings.

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

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

Negation

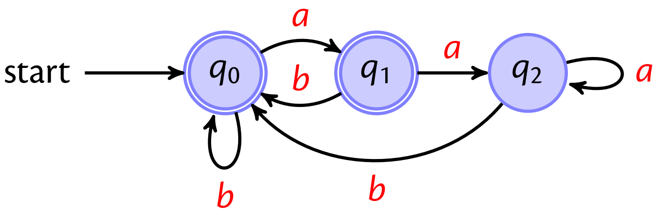
Regular languages are closed under negation:



But requires that the automaton is completed!

Negation

Regular languages are closed under negation:



But requires that the automaton is completed!

Housekeeping

The deadline for CW2 is 6 November (thanks to Arshdeep Pareek for pointing this out).

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?