Automata and Formal Languages (3)

Email: christian.urban at kcl.ac.uk
Office: S1.27 (1st floor Strand Building)

Slides: KEATS (also home work and coursework is there)

Regular Expressions

In programming languages they are often used to recognise:

- symbols, digits
- identifiers
- numbers (non-leading zeros)
- keywords
- comments

http://www.regexper.com

Last Week

Last week I showed you a regular expression matcher which works provably correctly in all cases.

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

by Janusz Brzozowski (1964)

The Derivative of a Rexp

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

To see what is going on, define

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

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

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

The Idea of the Algorithm

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

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

The Idea of the Algorithm

If we want to recognise the string $\textit{"abc"}$ with regular expression r then

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

The Idea of the Algorithm

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

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

The Idea of the Algorithm

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

1. $Der\, a\, (L(r))$
2. $Der\, b\, (Der\, a\, (L(r)))$
3. $Der\, c\, (Der\, b\, (Der\, a\, (L(r))))$
4. finally we test whether the empty string is in this set

The Idea of the Algorithm

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

1. $Der\,a\,(L(r))$
2. $Der\,b\,(Der\,a\,(L(r)))$
3. $Der\,c\,(Der\,b\,(Der\,a\,(L(r))))$
4. finally we test whether the empty string is in this set

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

Input: string "abc" and regular expression r

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

Input: string "abc" and regular expression r

1. $der\ a\ r$
2. $der\ b\ (der\ a\ r)$
3. $der\ c\ (der\ b\ (der\ a\ r))$

4. finally check whether the last regular expression can match the empty string

We proved already

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

by induction on the regular expression.

We proved already

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

by induction on the regular expression.

Any Questions?

We need to prove

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

by induction on the regular expression.

Proofs about Rexps

- P holds for $\varnothing$, ϵ and c
- P holds for $r_1 + r_2$ under the assumption that P already holds for r_1 and r_2.
- P holds for $r_1 \cdot r_2$ under the assumption that P already holds for r_1 and r_2.
- P holds for r^* under the assumption that P already holds for r.

Proofs about Natural Numbers and Strings

- P holds for 0 and
- P holds for $n + 1$ under the assumption that P already holds for n

- P holds for "" and
- P holds for $c :: s$ under the assumption that P already holds for s

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.

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. $a^n b^n$.

Regular Expressions

$$
\begin{array}{rcll}
r & ::= & \varnothing & \text{null} \\
 & \mid & \epsilon & \text{empty string / "" / []} \\
 & \mid & c & \text{character} \\
 & \mid & r_1 \cdot r_2 & \text{sequence} \\
 & \mid & r_1 + r_2 & \text{alternative / choice} \\
 & \mid & r^* & \text{star (zero or more)}
\end{array}
$$

How about ranges $[a\text{-}z]$, r^+ and $\sim r$? Do they increase the set of languages we can recognise?

Negation of Regular Expr's

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

Negation of Regular Expr's

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

Used often for recognising comments:

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

Negation

Assume you have an alphabet consisting of the letters a, b and c only. Find a regular expression that matches all strings except ab and ac.

Regular Exp's for Lexing

Lexing separates strings into "words" / components.

- Identifiers (non-empty strings of letters or digits, starting with a letter)
- Numbers (non-empty sequences of digits omitting leading zeros)
- Keywords (else, if, while, ...)
- White space (a non-empty sequence of blanks, newlines and tabs)
- Comments

Automata

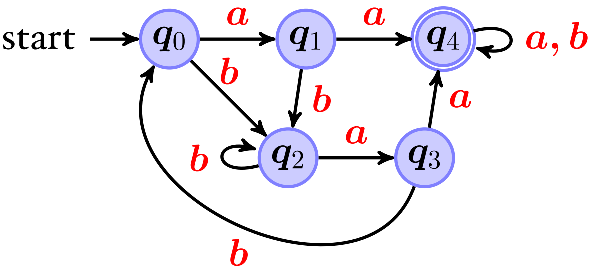
A deterministic finite automaton consists of:

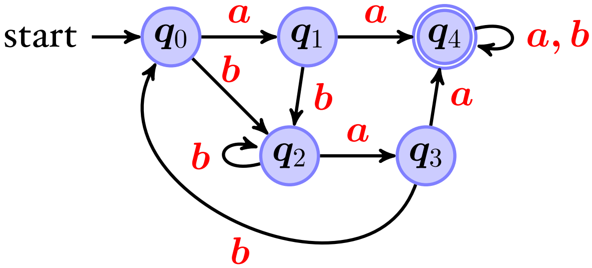
- a set of states
- one of these states is the start state
- some states are accepting states, and
- there is transition function

  which takes a state as argument and a character and produces a new state

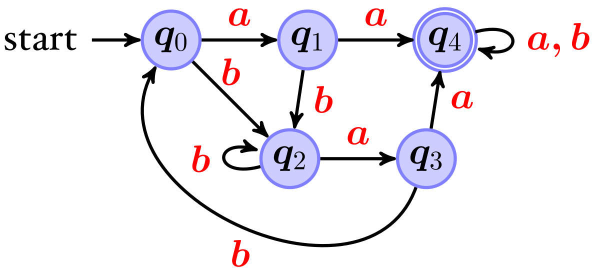
  this function might not be everywhere defined

$$A(Q, q_0, F, \delta)$$


start
q_0
q_1
q_2
q_3
q_4
a
a
b
b
a
a
b
b
a, b



- 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

$$A(Q, q_0, F, \delta)$$

you can define

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

Accepting a String

Given

$$A(Q, q_0, F, \delta)$$

you can define

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

Whether a string s is accepted by A?

$$\hat{\delta}(q_0, s) \in F$$

Non-Deterministic Finite Automata

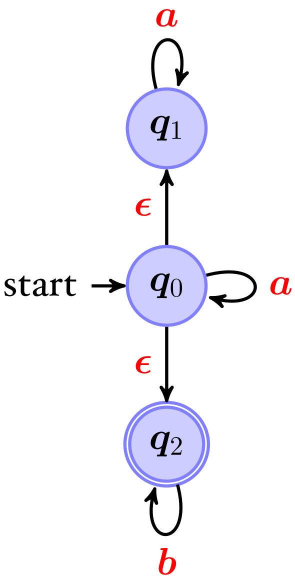
A non-deterministic finite automaton consists again of:

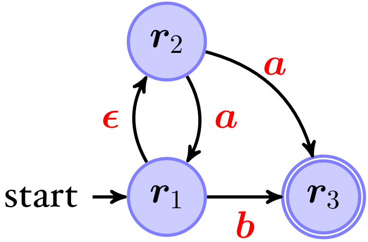
- a finite set of states
- one of these states is the start state
- some states are accepting states, and
- there is transition relation

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

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

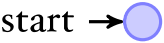
Two NFA Examples



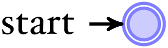


Rexp to NFA

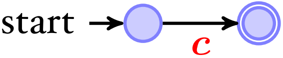
$\varnothing$



ϵ

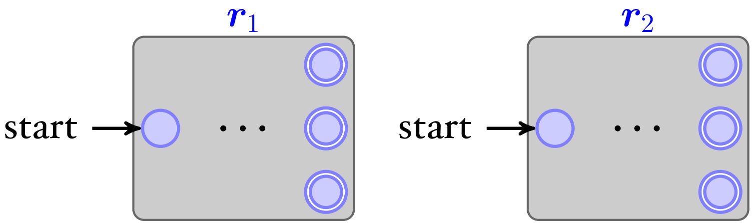


c



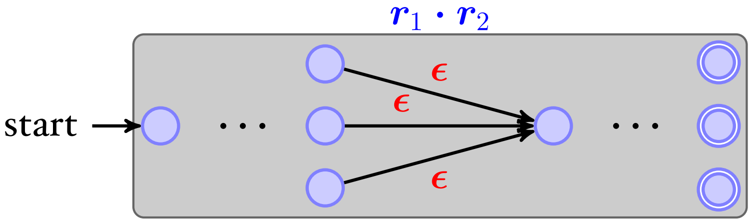
Case $r_1 \cdot r_2$

By recursion we are given two automata:



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

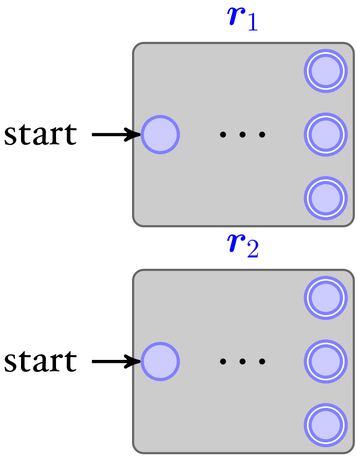
Case $r_1 \cdot r_2$



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

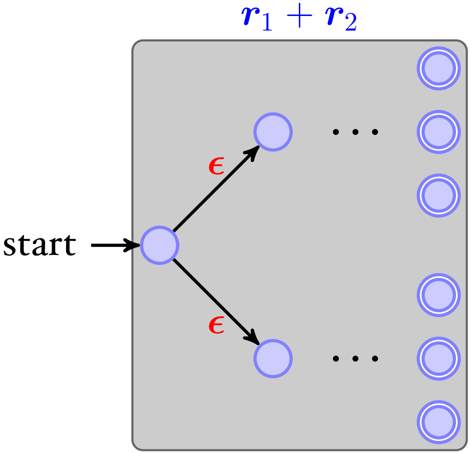
Case $r_1 + r_2$

By recursion we are given two automata:



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

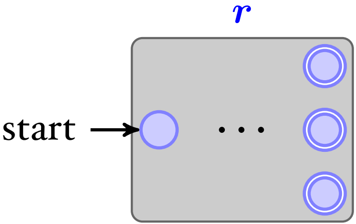
Case $r_1 + r_2$



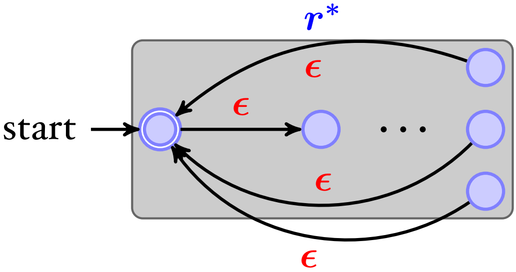
We (1) need to introduce a new starting state and (2) connect it to the original two starting states.

Case r^*

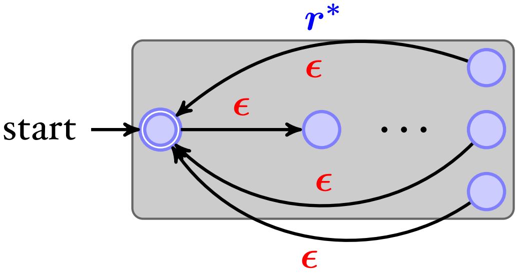
By recursion we are given an automaton for r:



Case r^*

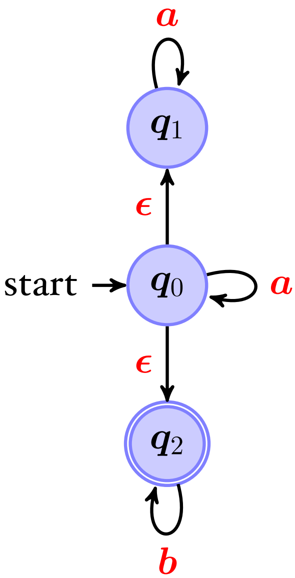


Case r^*



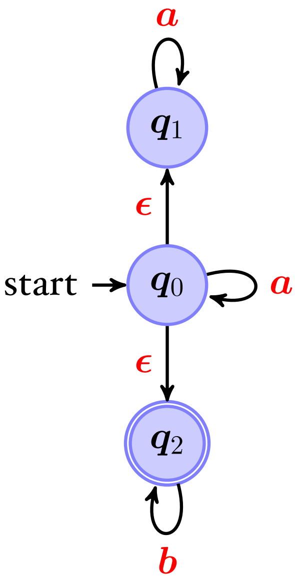
Why can't we just have an epsilon transition from the accepting states to the starting state?

Subset Construction



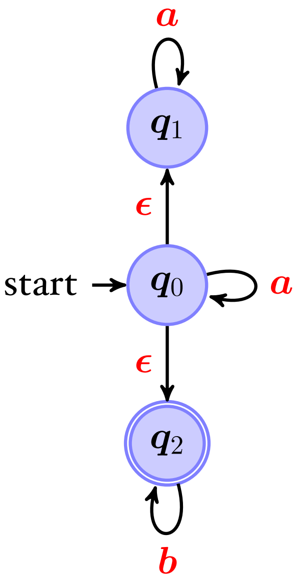
nodes	a	b
$\varnothing$		
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



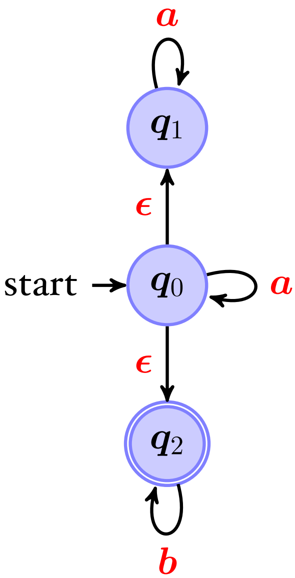
nodes	a	b
$\varnothing$	$\varnothing$	$\varnothing$
$\{0\}$		
$\{1\}$		
$\{2\}$		
$\{0,1\}$		
$\{0,2\}$		
$\{1,2\}$		
$\{0,1,2\}$		

Subset Construction



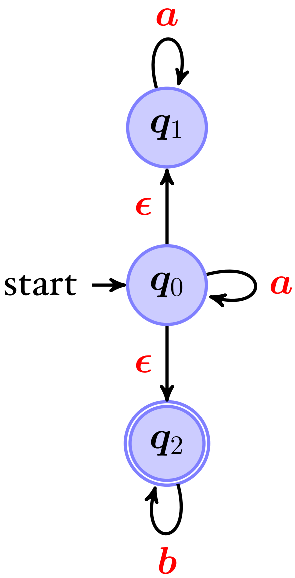
nodes	a	b
$\varnothing$	$\varnothing$	$\varnothing$
$\{0\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\varnothing$
$\{2\}$	$\varnothing$	$\{2\}$
$\{0, 1\}$		
$\{0, 2\}$		
$\{1, 2\}$		
$\{0, 1, 2\}$		

Subset Construction



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

Subset Construction



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

Regular Languages

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.

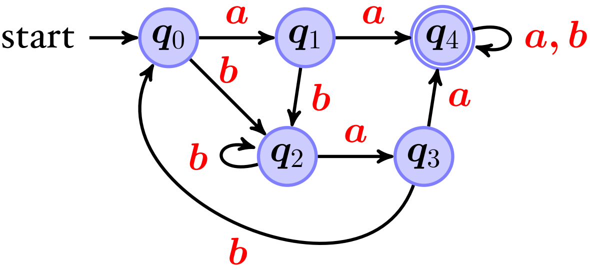
Regular Languages

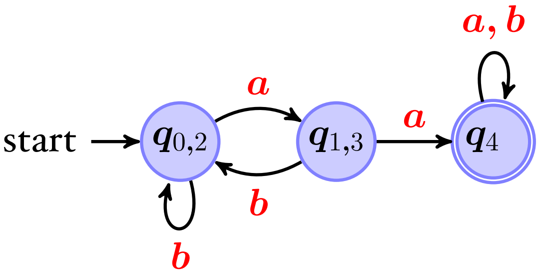
A language is regular iff there exists a regular expression that recognises all its strings.

or **equivalently**

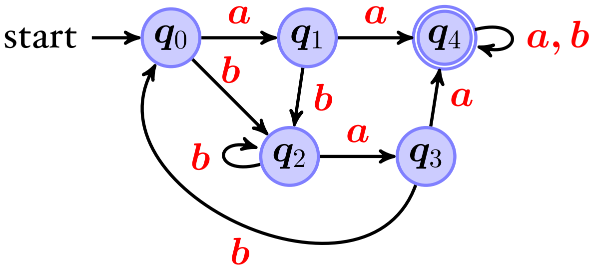
A language is regular iff there exists a deterministic finite automaton that recognises all its strings.

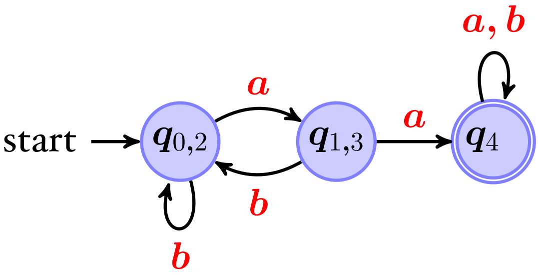
Why is every finite set of strings a regular language?





minimal automaton





minimal automaton

Given the function

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

and the set

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

prove whether

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