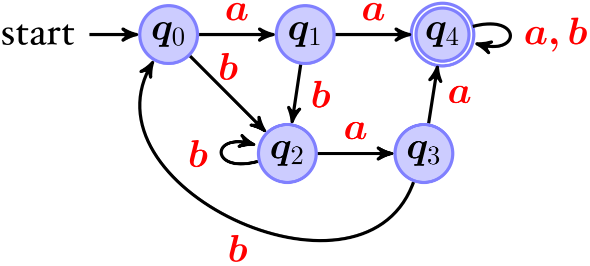
Automata and Formal Languages (5)

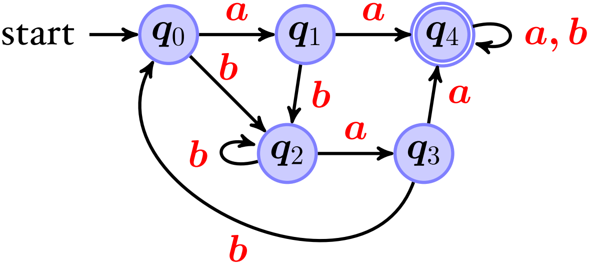
Email: christian.urban at kcl.ac.uk
Office: S1.27 (1st floor Strand Building)
Slides: KEATS (also home work is there)

DFA Minimisation

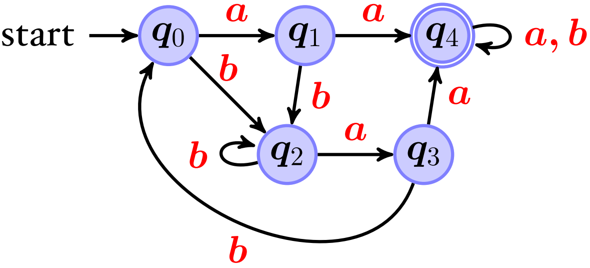
1. Take all pairs (q, p) with $q \neq p$
2. Mark all pairs that accepting and non-accepting states
3. For all unmarked pairs (q, p) and all characters c tests wether
$$(\delta(q, c), \delta(p, c))$$
are marked. If yes, then also mark (q, p).
4. Repeat last step until no chance.
5. All unmarked pairs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



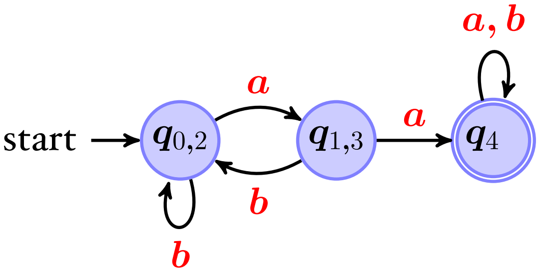
q_1				
q_2				
q_3				
q_4	⋆	⋆	⋆	⋆
	q_0	q_1	q_2	q_3



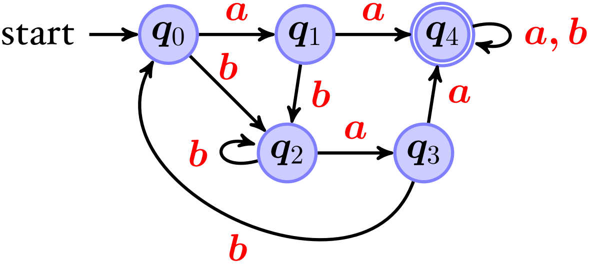
	q_0	q_1	q_2	q_3
q_1				
q_2				
q_3				
q_4	$\star$	$\star$	$\star$	$\star$



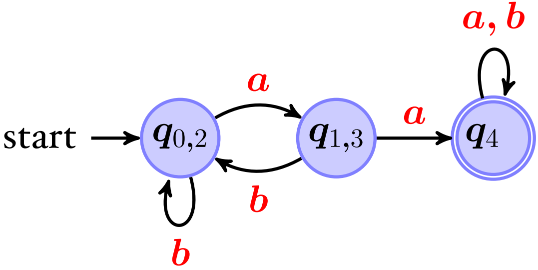
	q_0	q_1	q_2	q_3
q_1	★			
q_2		★		
q_3	★		★	
q_4	★	★	★	★



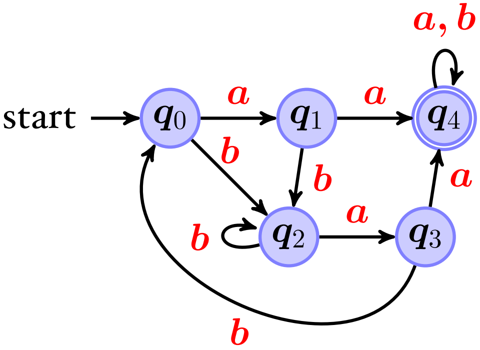
minimal automaton

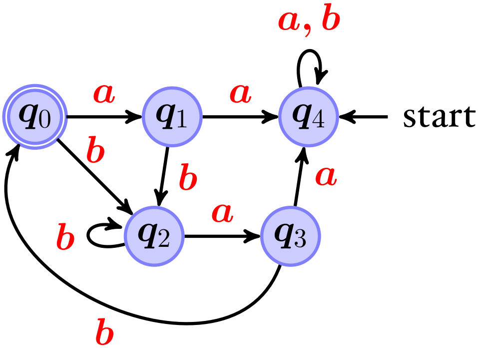


	q_0	q_1	q_2	q_3
q_1	★			
q_2		★		
q_3	★		★	
q_4	★	★	★	★

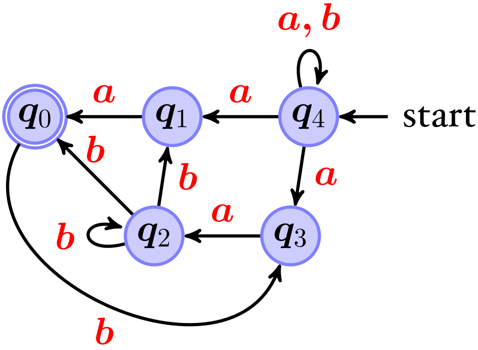


minimal automaton

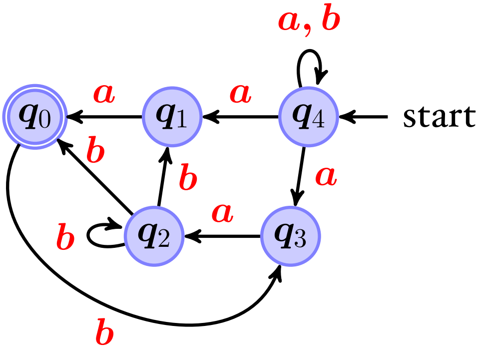

a, b
start
q_0
a
q_1
a
q_4
b
b
a
b
q_2
a
q_3
b



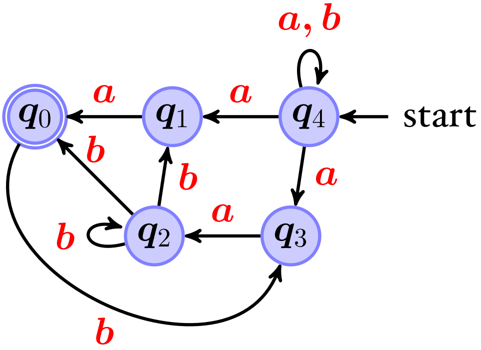
- exchange initial / accepting states



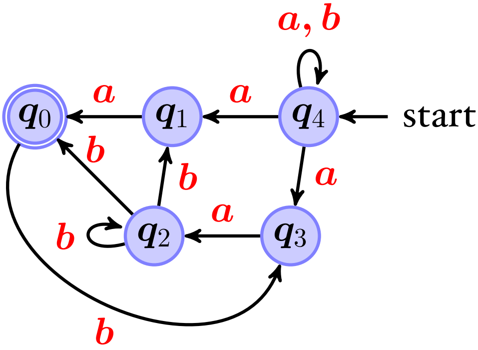
- exchange initial / accepting states
- reverse all edges



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



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

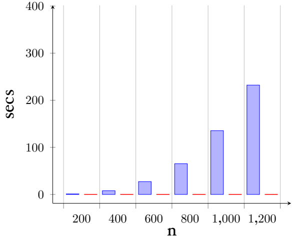


- exchange initial / accepting states
- reverse all edges
- subset construction ⇒ DFA
- repeat once more ⇒ minimal DFA

```
write ”Input a number ”;
read n;
x := 0;
y := 1;
while n > 0 do {
  temp := y;
  y := x + y;
  x := temp;
  n := n - 1
};
write ”Result ”;
write y
```

```
write ”Input a number ”;
read n;
while n > 1 do {
  if n % 2 == 0
  then n := n/2
  else n := 3*n+1;
};
write ”Yes”;
```

```
start := 1000;
x := start;
y := start;
z := start;
while 0 < x do {
 while 0 < y do {
  while 0 < z do { z := z - 1 };
  z := start;
  y := y - 1
 };
 y := start;
 x := x - 1
};
```



”if true then then 42 else +”

KEYWORD:

if, then, else,

WHITESPACE:

” ”, \n,

IDENT:

$LETTER \cdot (LETTER + DIGIT + \_)^*$

NUM:

$(NONZERODIGIT \cdot DIGIT^*) + 0$

OP:

+

COMMENT:

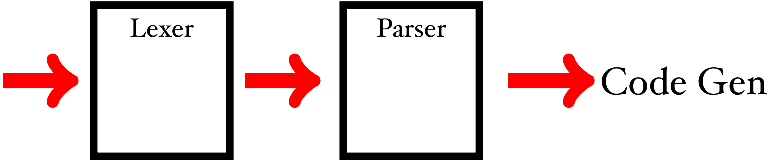
$/* \cdot (ALL^* \cdot */ \cdot ALL^*) \cdot */$

”if true then then 42 else +”

```
KEYWORD(if),
WHITESPACE,
IDENT(true),
WHITESPACE,
KEYWORD(then),
WHITESPACE,
KEYWORD(then),
WHITESPACE,
NUM(42),
WHITESPACE,
KEYWORD(else),
WHITESPACE,
OP(+)
```

”if true then then 42 else +”

```
KEYWORD(if),
IDENT(true),
KEYWORD(then),
KEYWORD(then),
NUM(42),
KEYWORD(else),
OP(+)
```


Lexer
Parser
Code Gen

Two Rules

- Longest match rule ("maximal munch rule"): The longest initial substring matched by any regular expression is taken as next token.

- Rule priority: For a particular longest initial substring, the first regular expression that can match determines the token.

Nullable

...whether a regular expression can match the empty string:

$$
\begin{array}{lcl}
nullable(\varnothing) & \stackrel{\text{def}}{=} & false \\
nullable(\epsilon) & \stackrel{\text{def}}{=} & true \\
nullable(c) & \stackrel{\text{def}}{=} & false \\
nullable(r_1 + r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \vee nullable(r_2) \\
nullable(r_1 \cdot r_2) & \stackrel{\text{def}}{=} & nullable(r_1) \wedge nullable(r_2) \\
nullable(r^*) & \stackrel{\text{def}}{=} & true
\end{array}
$$

Zeroable

...whether a regular expression can match nothing:

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

Grammars

$$
\begin{array}{lcl}
E & \rightarrow & F + (F \cdot \text{"}*\text{"} \cdot F) + (F \cdot \text{"}\backslash\text{"} \cdot F) \\
F & \rightarrow & T + (T \cdot \text{"+"} \cdot T) + (T \cdot \text{"-"} \cdot T) \\
T & \rightarrow & num + (\text{"("} \cdot E \cdot \text{")"})
\end{array}
$$

E, F and T are non-terminals
E is start symbol
num, (,), + ...are terminals

(2*3)+(3+4)

$$\begin{array}{lcl} \boldsymbol{E} & \rightarrow & \boldsymbol{F} + (\boldsymbol{F} \cdot \text{"} * \text{"} \cdot \boldsymbol{F}) + (\boldsymbol{F} \cdot \text{"}\backslash\text{"} \cdot \boldsymbol{F}) \\ \boldsymbol{F} & \rightarrow & \boldsymbol{T} + (\boldsymbol{T} \cdot \text{"+"} \cdot \boldsymbol{T}) + (\boldsymbol{T} \cdot \text{"-"} \cdot \boldsymbol{T}) \\ \boldsymbol{T} & \rightarrow & \boldsymbol{num} + (\text{"("} \cdot \boldsymbol{E} \cdot \text{")"}) \end{array}$$

(2*3)+(3+4)