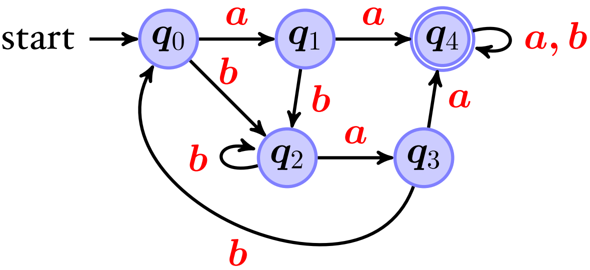
Automata and Formal Languages (5)

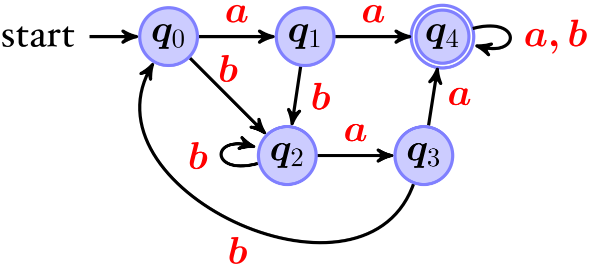
Email:	christian.urban at kcl.ac.uk
Office:	S1.27 (1st floor Strand Building)
Slides:	KEATS (also home work is there)

DFA Minimisation

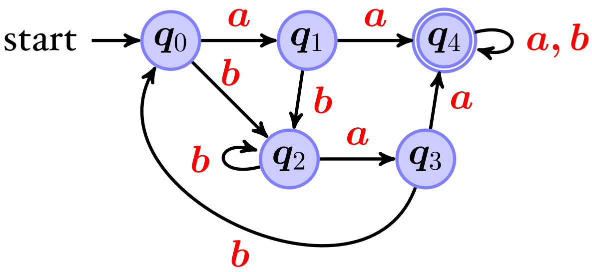
1. Take all pairs (q, p) with $q \neq p$
2. Mark all pairs that accepting and non-accepting states
3. For all unmarked pairs (q, p) and all characters c tests wether
$$(\delta(q, c), \delta(p, c))$$
are marked. If yes, then also mark (q, p).
4. Repeat last step until no chance.
5. All unmarked pairs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



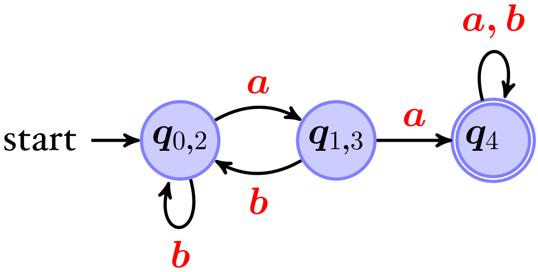
q_1				
q_2				
q_3				
q_4	$\star$	$\star$	$\star$	$\star$
	q_0	q_1	q_2	q_3



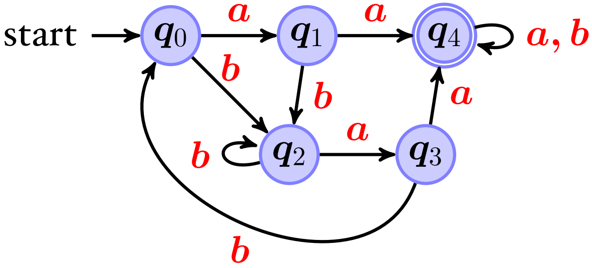
	q_0	q_1	q_2	q_3
q_1				
q_2				
q_3				
q_4	★	★	★	★



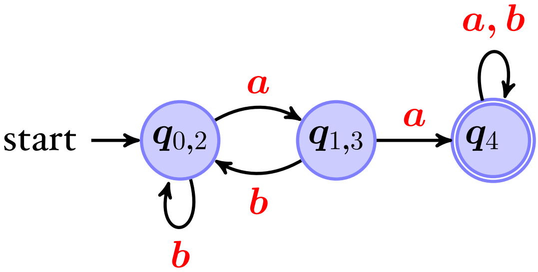
	q_0	q_1	q_2	q_3
q_1	★			
q_2		★		
q_3	★		★	
q_4	★	★	★	★



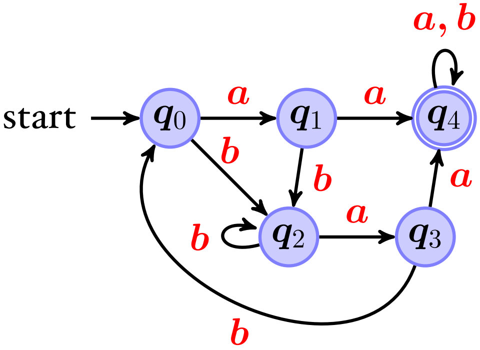
minimal automaton

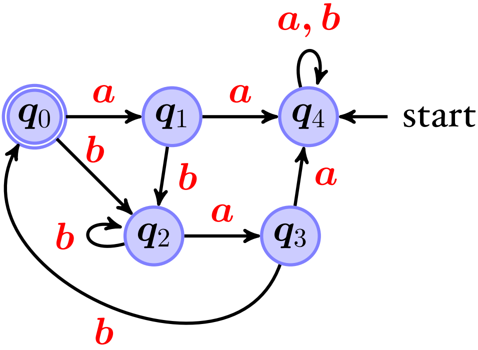


	q_0	q_1	q_2	q_3
q_1	★			
q_2		★		
q_3	★		★	
q_4	★	★	★	★

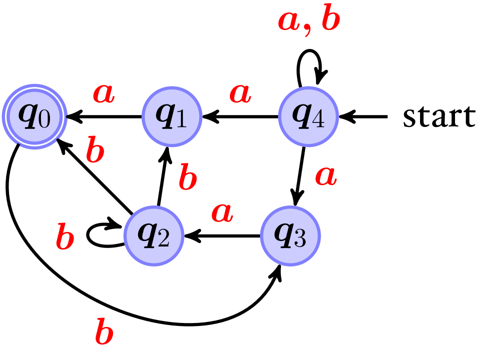


minimal automaton

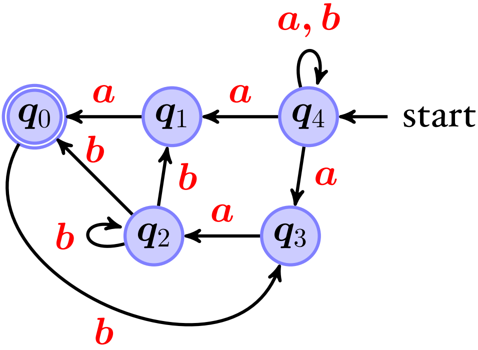

start
q_0
q_1
q_2
q_3
q_4
a
a
b
b
b
a
a
b
a, b



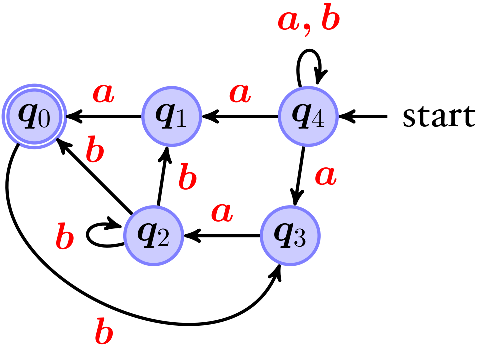
- exchange initial / accepting states



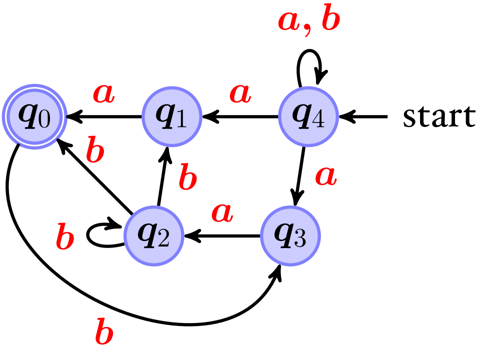
- exchange initial / accepting states
- reverse all edges



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



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



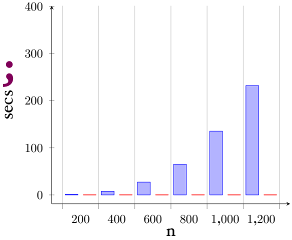
- exchange initial / accepting states
- reverse all edges
- subset construction ⇒ DFA
- repeat once more ⇒ minimal DFA

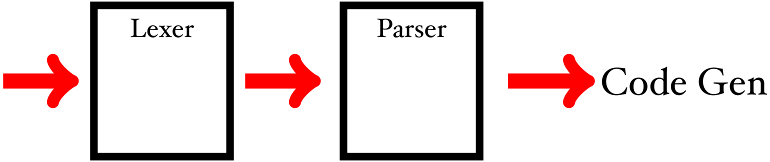
```
/* Fibonacci Program
   input: n */

write "Fib";
read n;    // n := 19;
minus1 := 0;
minus2 := 1;
while n > 0 do {
       temp := minus2;
       minus2 := minus1 + minus2;
       minus1 := temp;
       n := n - 1
};
write "Result";
write minus2
```

```
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”;
```

```
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";
```




Lexer
Parser
Code Gen

”if true then then 42 else +”

KEYWORD:
  if, then, else,
WHITESPACE:
  ” ”, \n,
IDENT:
  LETTER · (LETTER + DIGIT + _)*
NUM:
  (NONZERODIGIT · DIGIT*) + 0
OP:
  +
COMMENT:
  /* · (ALL* · */ · ALL*) · */

”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(+)
```

There is one small problem with the tokenizer.
How should we tokenize:

”x - 3”

```
OP:
  ”+”, ”-”
NUM:
  (NONZERODIGIT · DIGIT*) + ”0”
NUMBER:
  NUM + (”-” · NUM)
```

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:

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

Zeroable

...whether a regular expression can match nothing:

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

$$zeroable(r) \Leftrightarrow L(r) = \varnothing$$

- The star-case in our proof about the matcher needs the following lemma

$$Der\, c\, A^* = (Der\, c\, A) \mathbin{@} A^*$$

- $A^* = \{\text{""}\} \cup A \mathbin{@} A^*$
- If $\text{""} \in A$, then
  $Der\, c\, (A @ B) = (Der\, c\, A) @ B \cup (Der\, c\, B)$
- If $\text{""} \not\in A$, then
  $Der\, c\, (A @ B) = (Der\, c\, A) @ B$