Automata and Formal Languages (5)

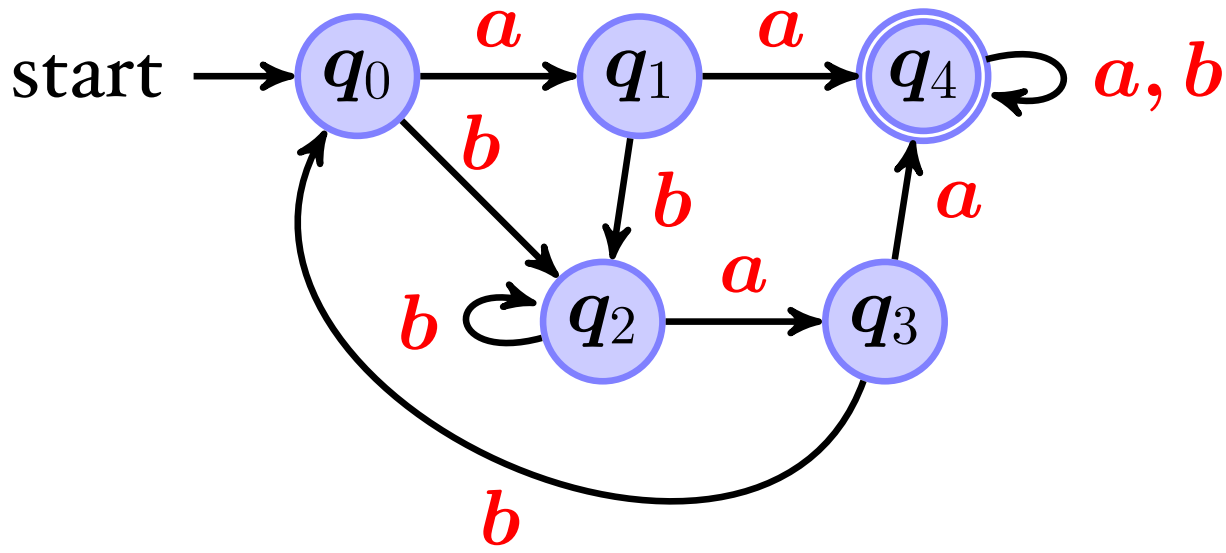
Email: christian.urban at kcl.ac.uk
Office: S1.27 (1st floor Strand Building)
Slides: KEATS (also home work is there)

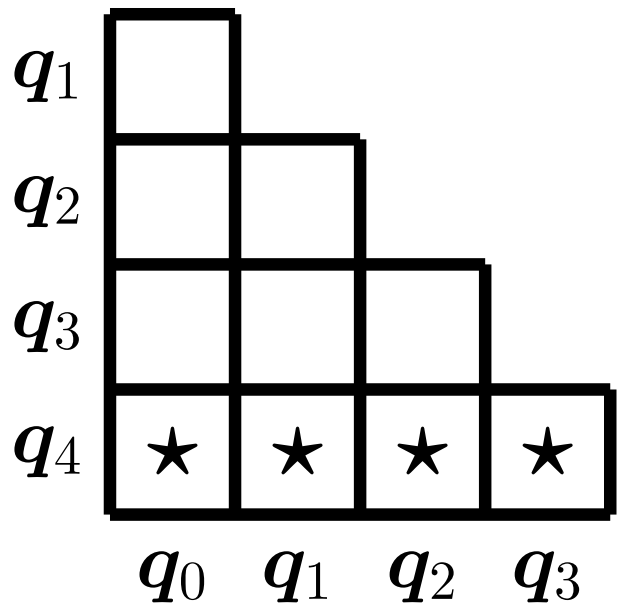
DFA Minimisation

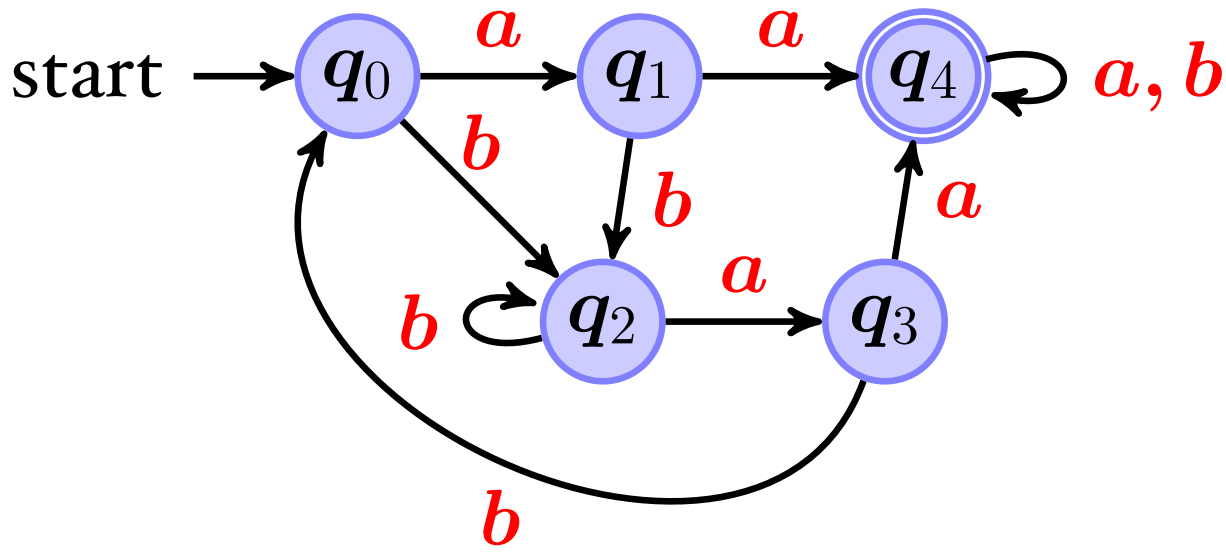
1. Take all pairs (q, p) with $q \neq p$
2. Mark all pairs that accepting and non-accepting states
3. For all unmarked pairs (q, p) and all characters c tests wether

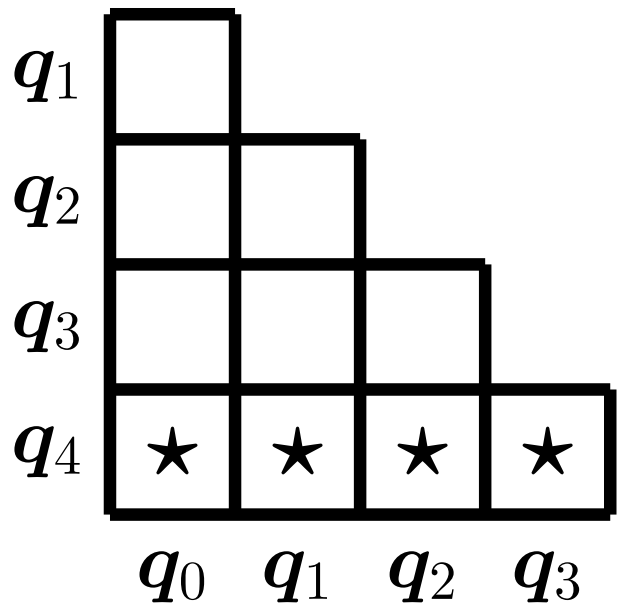
   $$(\delta(q, c), \delta(p, c))$$

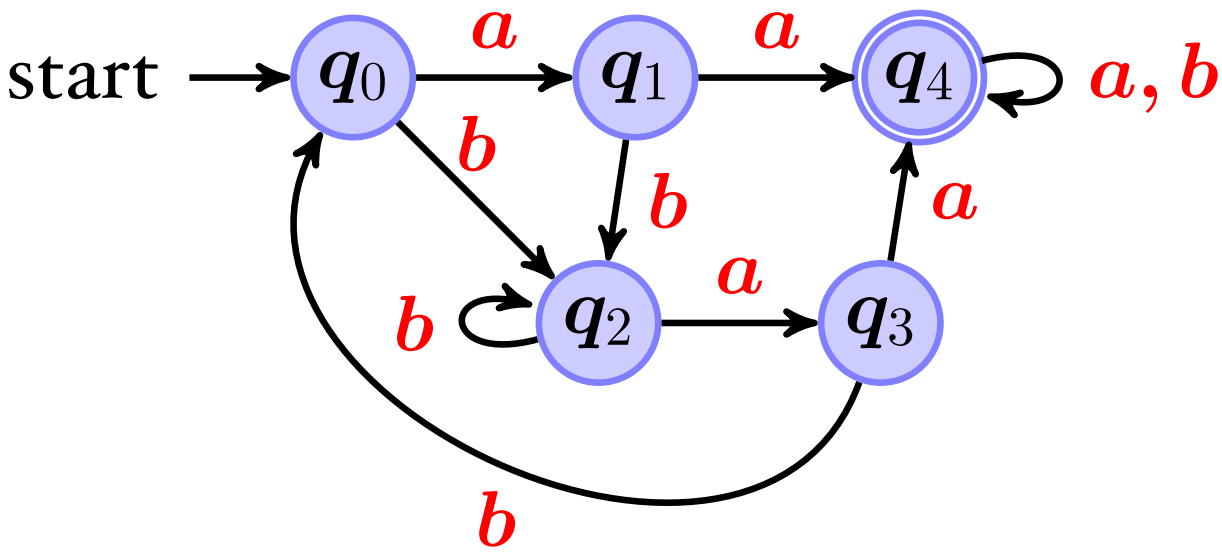
   are marked. If yes, then also mark (q, p).
4. Repeat last step until no chance.
5. All unmarked pairs can be merged.

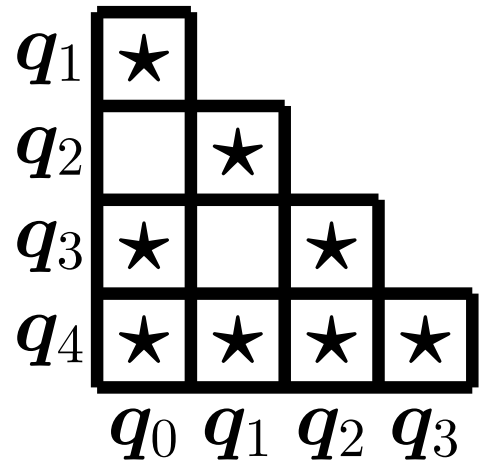

start
q_0
q_1
q_4
q_2
q_3
a
a
a, b
b
b
a
b
a
b

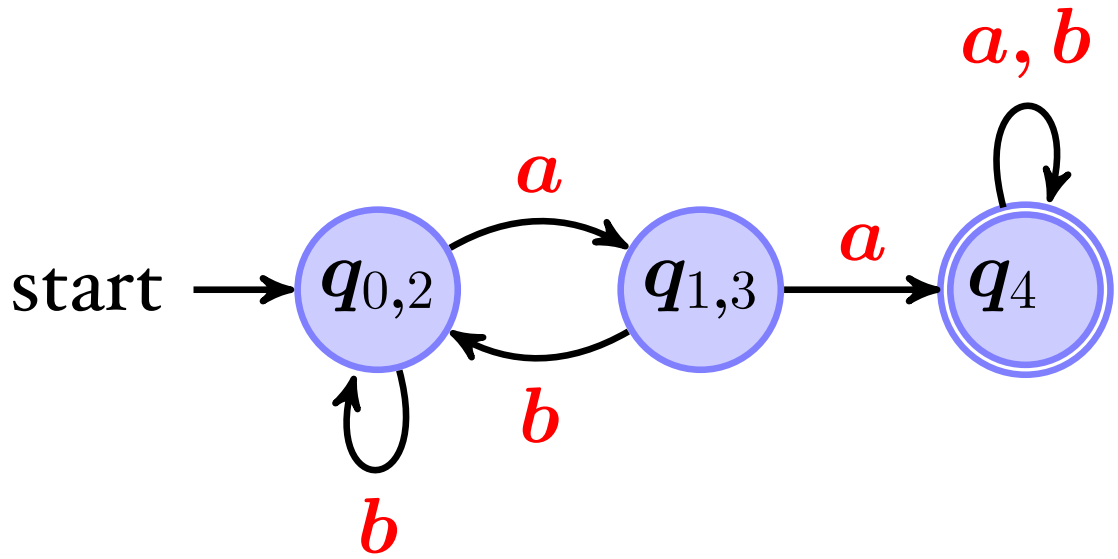

q_1
q_2
q_3
q_4
★
★
★
★
q_0
q_1
q_2
q_3


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

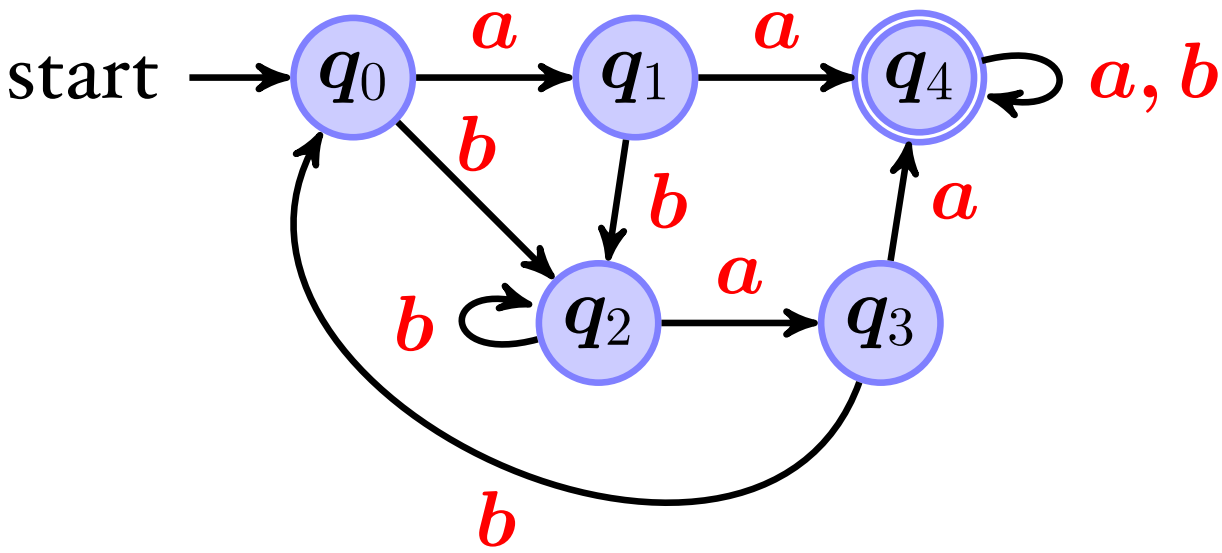

q_1
q_2
q_3
q_4
★
★
★
★
q_0
q_1
q_2
q_3

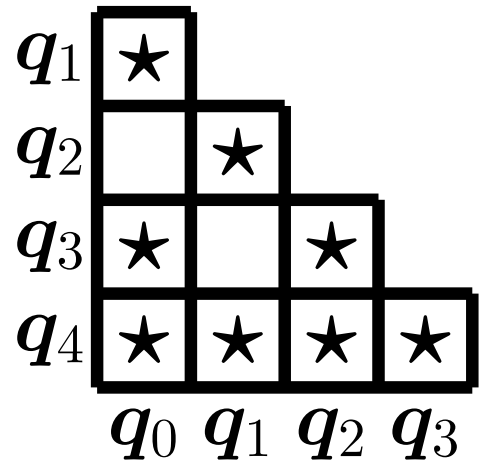


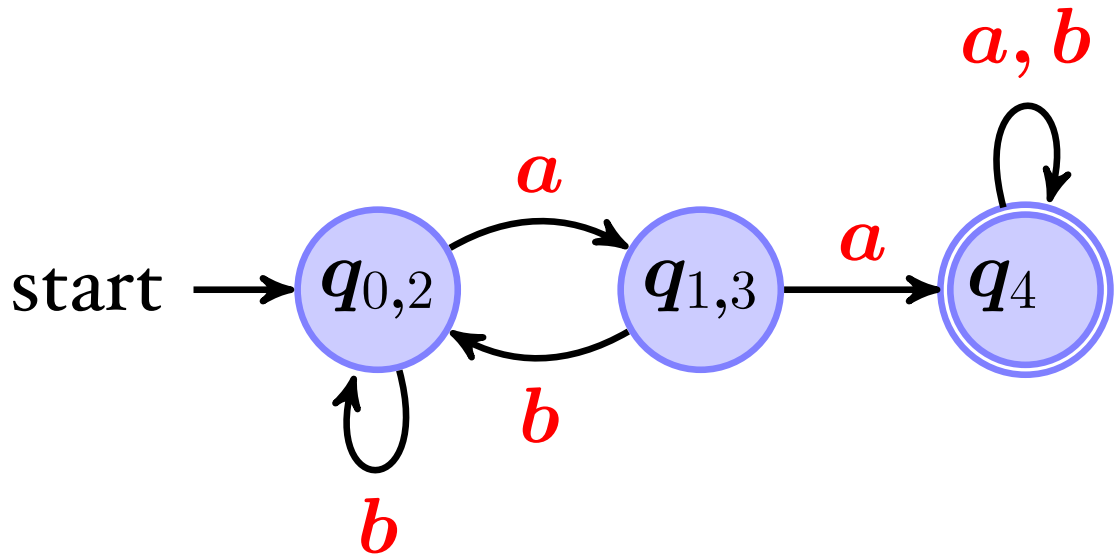




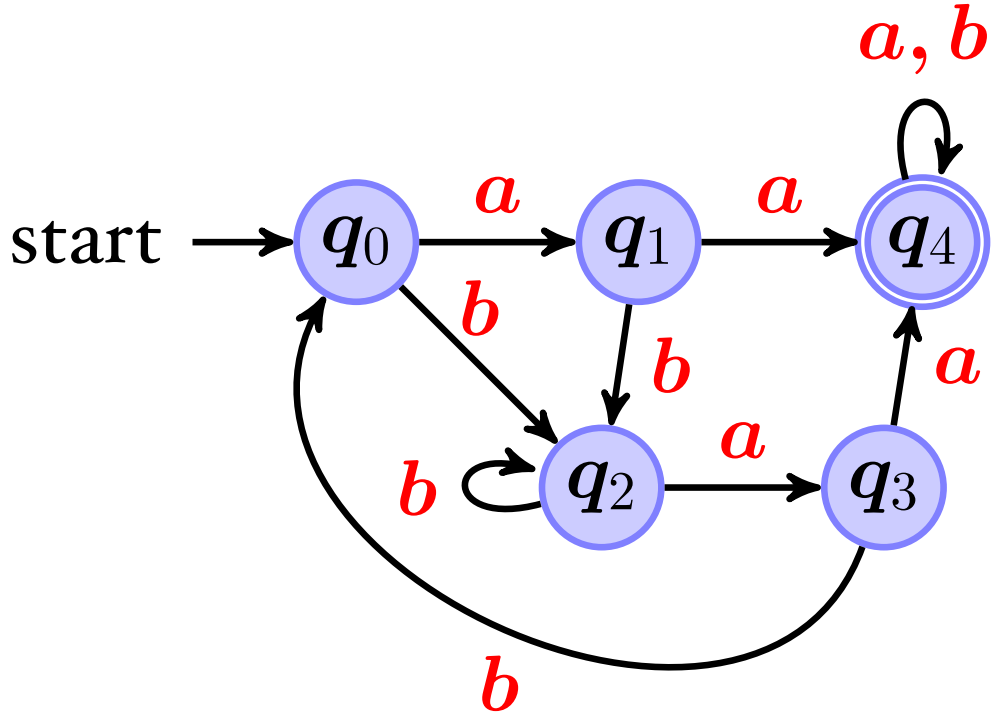
minimal automaton

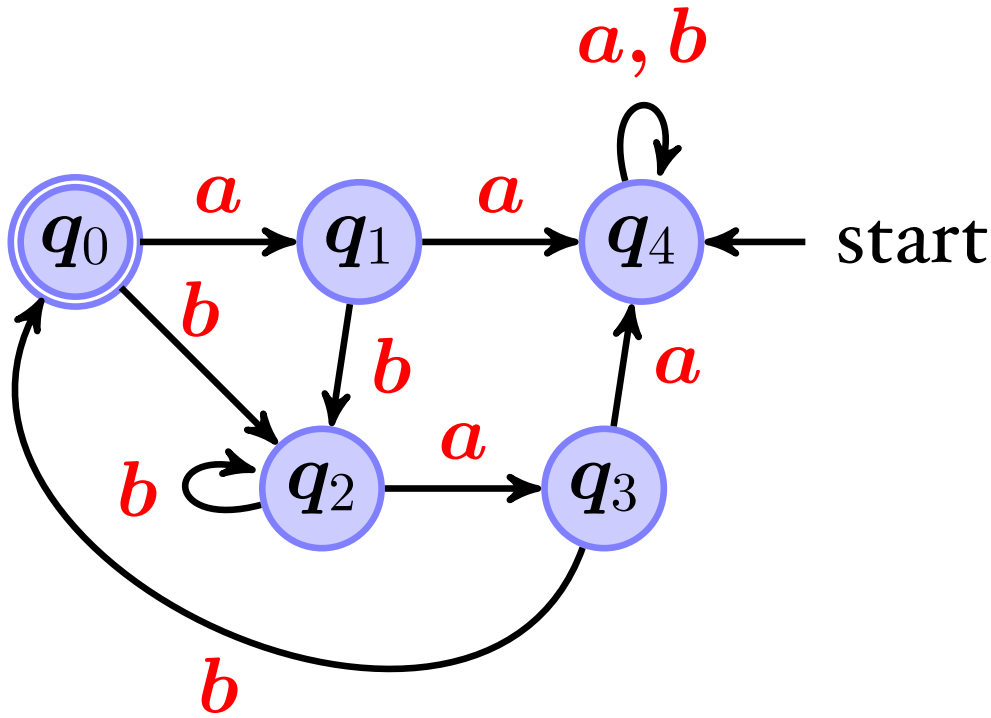




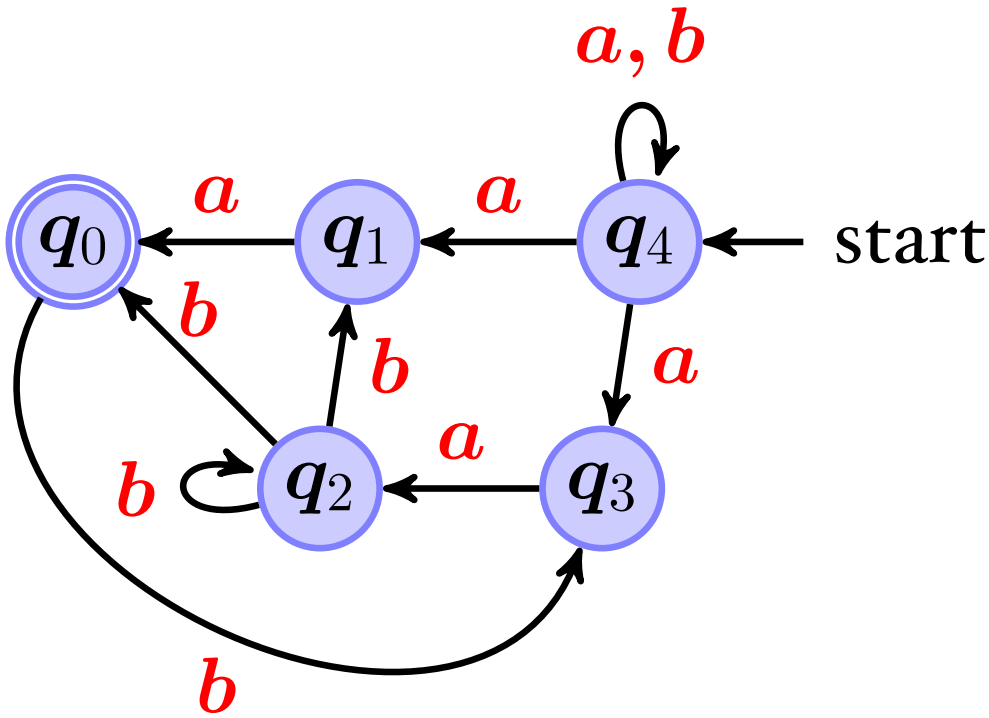


minimal automaton

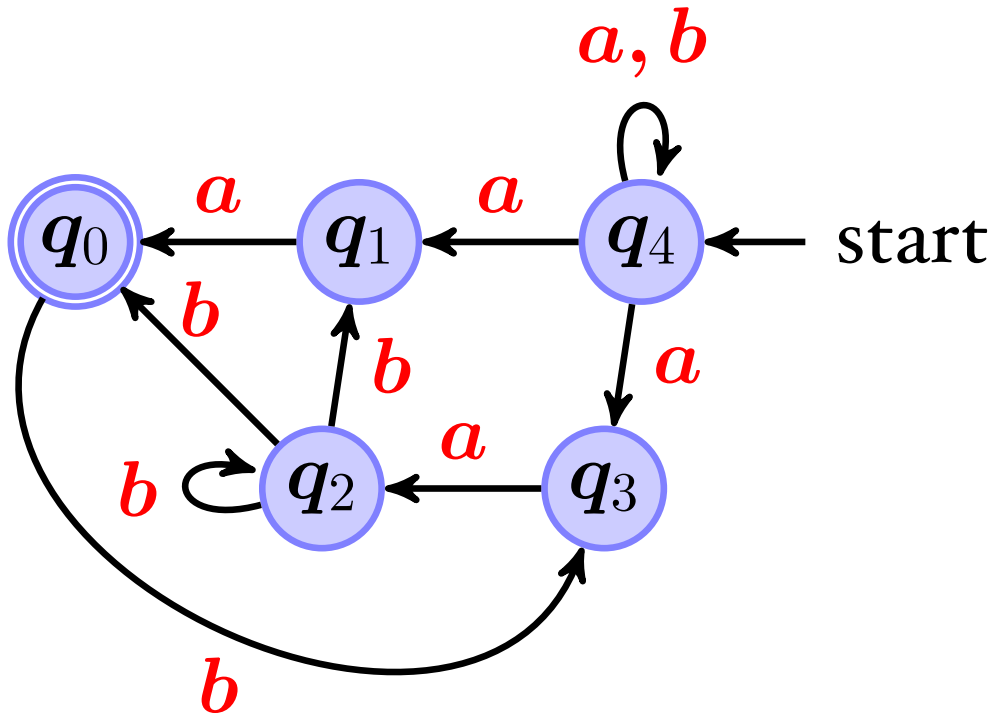

a, b
start
q0
a
q1
a
q4
b
b
a
b
q2
a
q3
b



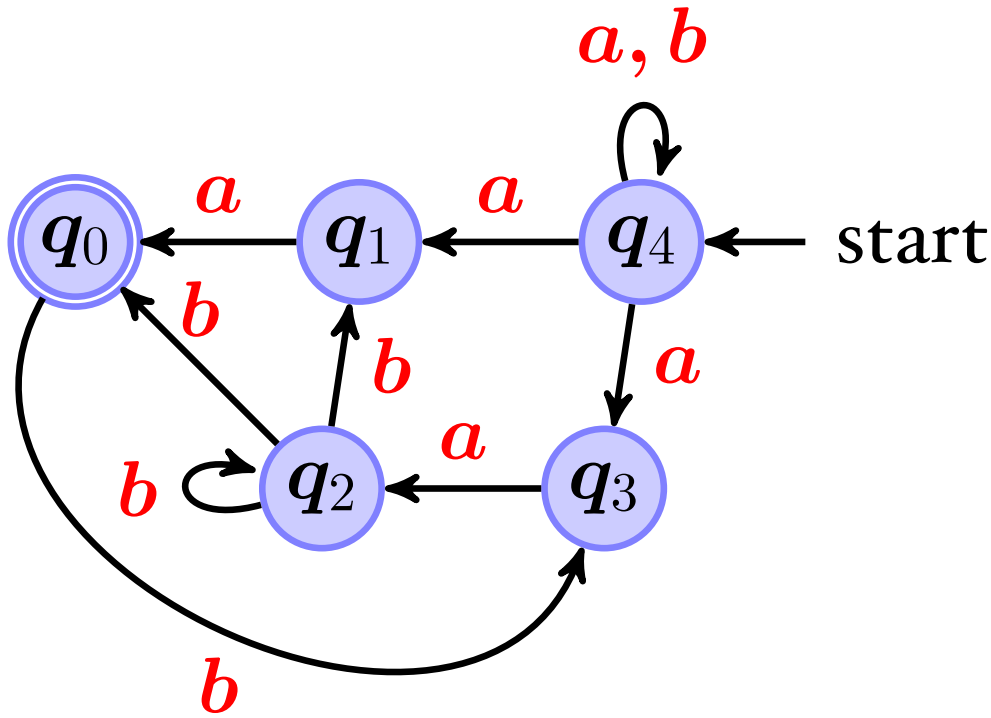
- exchange initial / accepting states



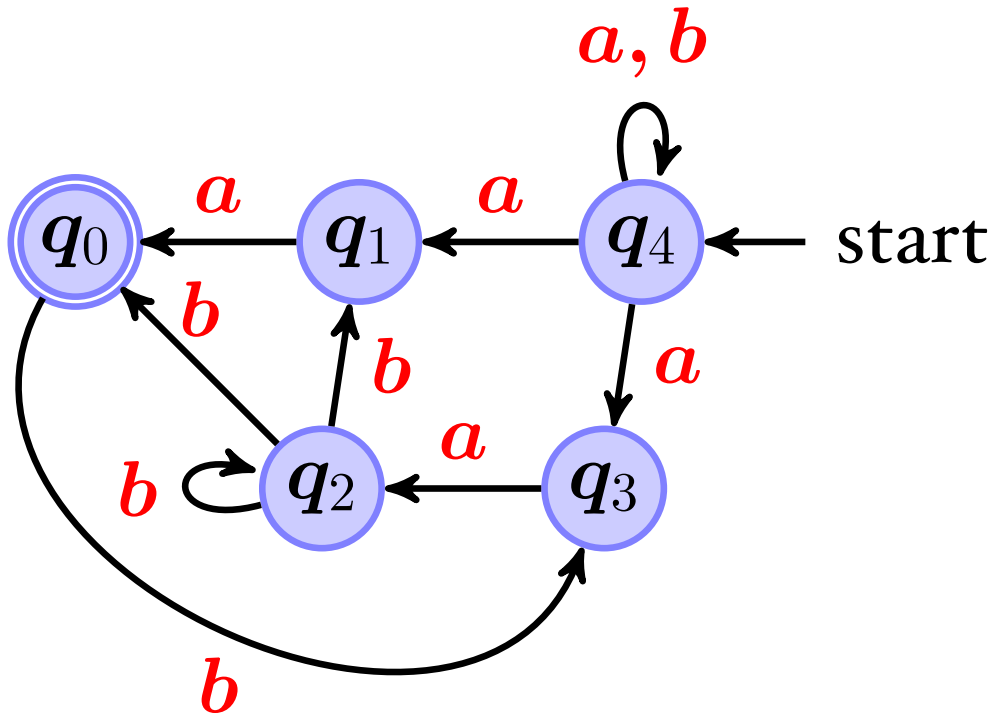
- exchange initial / accepting states
- reverse all edges



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



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

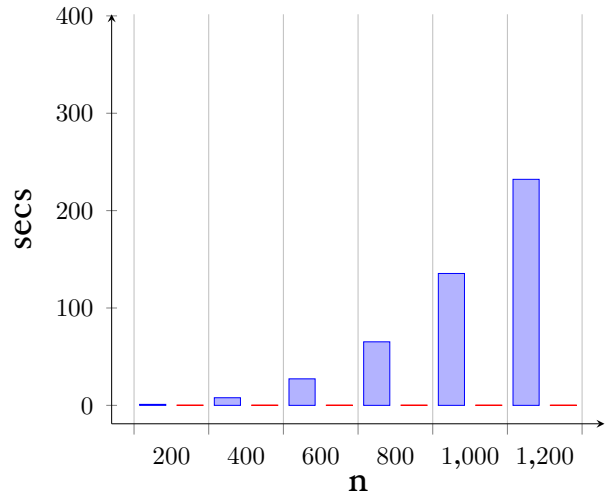


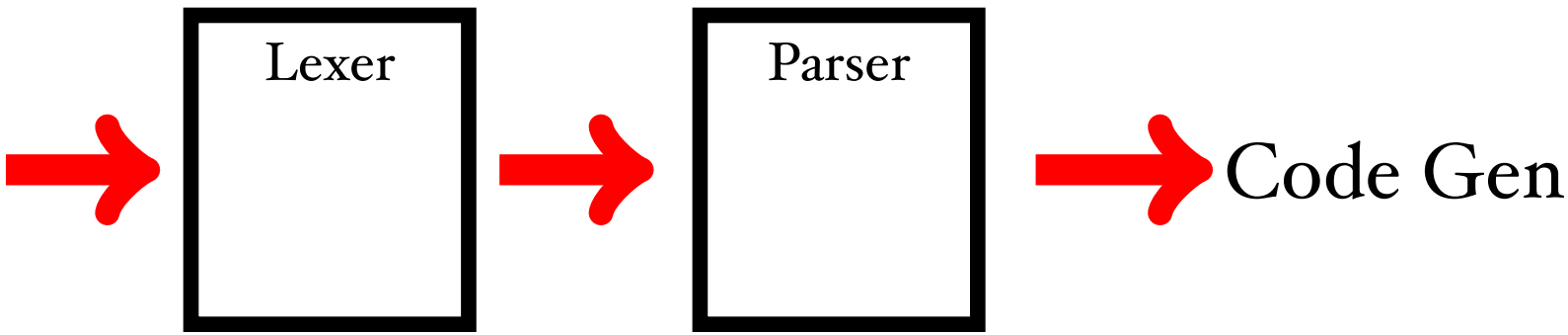
- exchange initial / accepting states
- reverse all edges
- subset construction $\Rightarrow$ DFA
- repeat once more $\Rightarrow$ minimal DFA

```
1  write "Input a number ";
2  read n;
3  x := 0;
4  y := 1;
5  while n > 0 do {
6     temp := y;
7     y := x + y;
8     x := temp;
9     n := n - 1
10 };
11 write "Result ";
12 write y
```

```
1 write ”Input a number ”;
2 read n;
3 while n > 1 do {
4   if n % 2 == 0
5   then n := n/2
6   else n := 3*n+1;
7 };
8 write ”Yes”;
```

```
1  start := 1000;
2  x := start;
3  y := start;
4  z := start;
5  while 0 < x do {
6   while 0 < y do {
7    while 0 < z do { z := z - 1 };
8    z := start;
9    y := y - 1
10  };
11  y := start;
12  x := x - 1
13 };
```




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:

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

Zeroable

...whether a regular expression can match nothing:

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

$$\boldsymbol{zeroable}(r) \Leftrightarrow \boldsymbol{L}(r) = \varnothing$$

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

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

- If $"" \in A$, then
$Der\, c\, (A @ B) = (Der\, c\, A) @ B \cup (Der\, c\, B)$
- If $"" \notin A$, then
$Der\, c\, (A @ B) = (Der\, c\, A) @ B$

Grammars

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

$\boldsymbol{E}$, $\boldsymbol{F}$ and $\boldsymbol{T}$ are non-terminals
$\boldsymbol{E}$ is start symbol
$\boldsymbol{num}$, (,), + ...are terminals

(2*3)+(3+4)

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

(2*3)+(3+4)