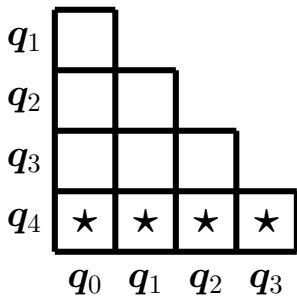
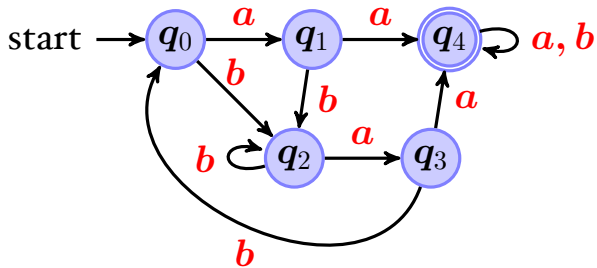


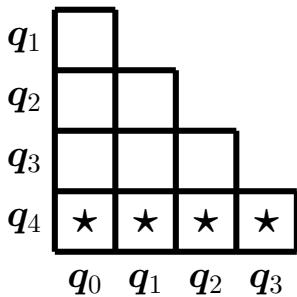
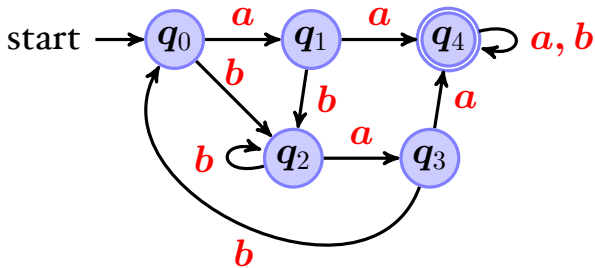
Automata and Formal Languages (5)

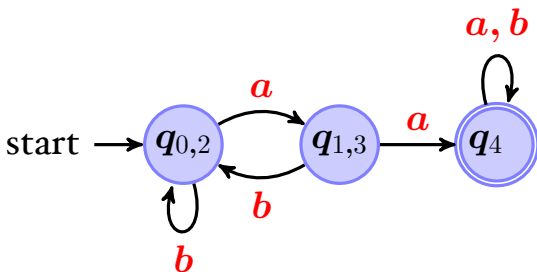
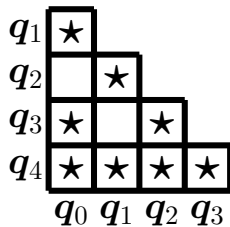
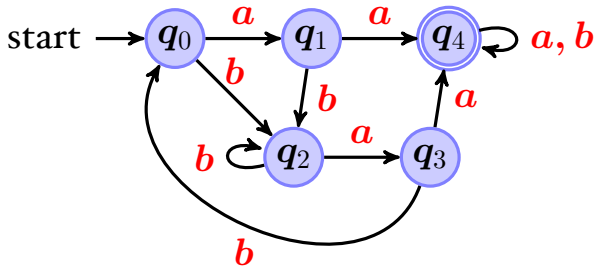
Email: christian.urban at kcl.ac.uk
Office: SI.27 (1st floor Strand Building)
Slides: KEATS (also home work is there)

DFA Minimisation

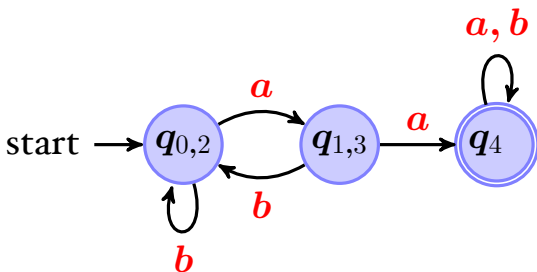
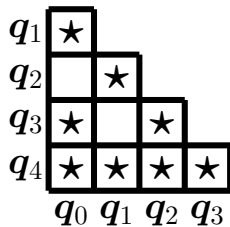
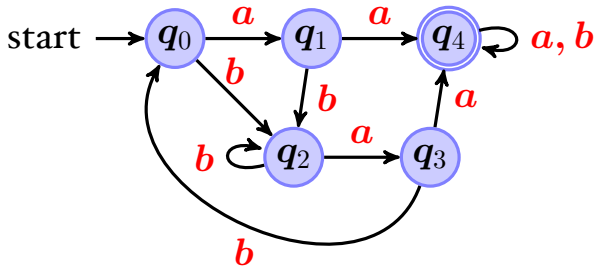
- 1 Take all pairs (q, p) with $q \neq p$
- 2 Mark all pairs that accepting and non-accepting states
- 3 For all unmarked pairs (q, p) and all characters c tests whether
$$(\delta(q, c), \delta(p, c))$$
are marked. If yes, then also mark (q, p) .
- 4 Repeat last step until no change.
- 5 All unmarked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



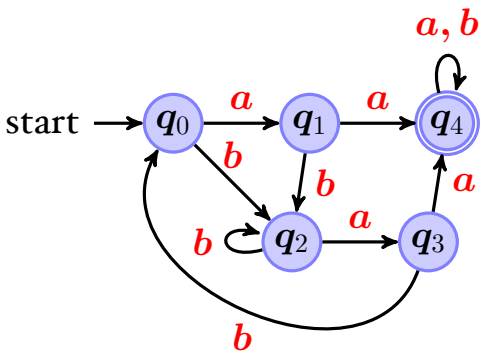


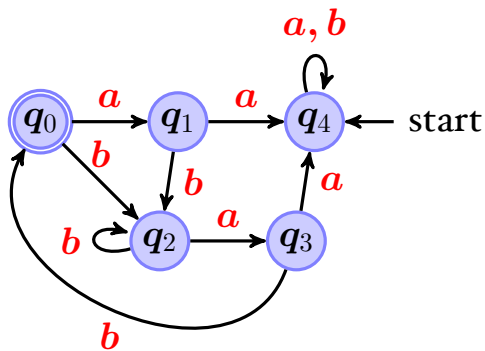


minimal automaton

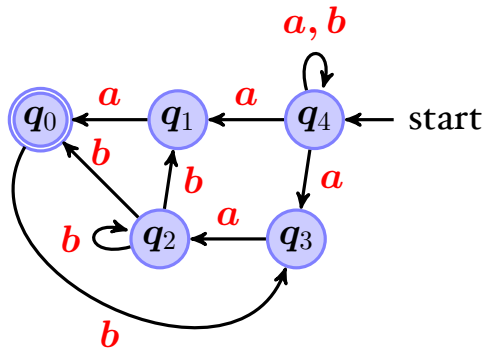


minimal automaton

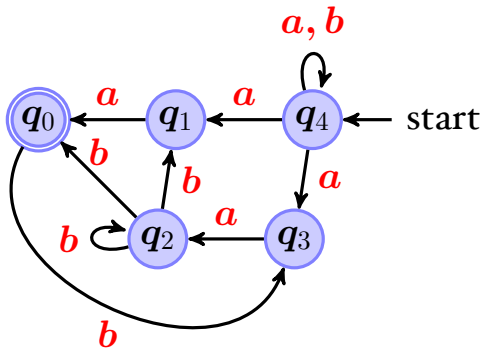




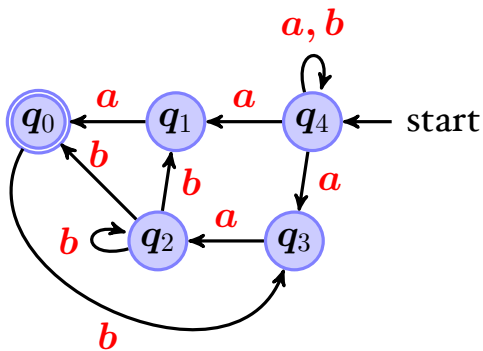
- exchange initial / accepting states



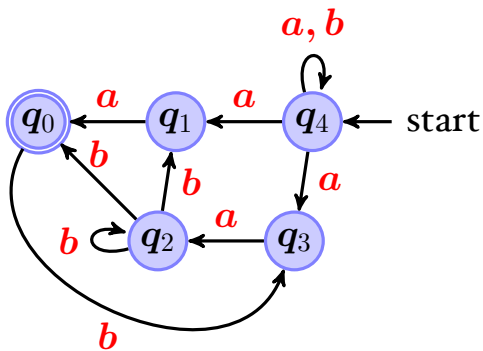
- exchange initial / accepting states
- reverse all edges



- 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

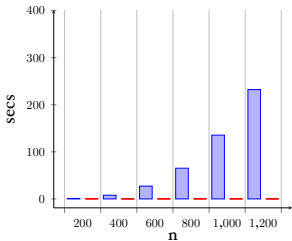
```
1 write "Input a number ";
2 read n;
3 x := 0;
4 y := 1;
5 while n > 0 do {
6     temp := y;
7     y := x + y;
8     x := temp;
9     n := n - 1
10 };
11 write "Result ";
12 write y
```

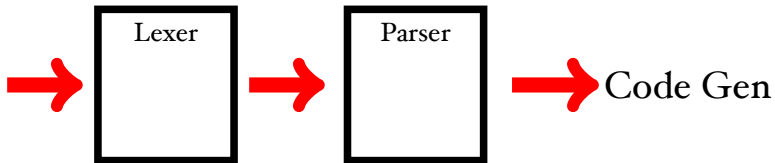
```
1 write "Input a number ";
2 read n;
3 while n > 1 do {
4     if n % 2 == 0
5     then n := n/2
6     else n := 3*n+1;
7 };
8 write "Yes";
```

```

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 };

```





”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:

$$\mathit{nullable}(\emptyset) \stackrel{\text{def}}{=} \mathit{false}$$

$$\mathit{nullable}(\epsilon) \stackrel{\text{def}}{=} \mathit{true}$$

$$\mathit{nullable}(c) \stackrel{\text{def}}{=} \mathit{false}$$

$$\mathit{nullable}(r_1 + r_2) \stackrel{\text{def}}{=} \mathit{nullable}(r_1) \vee \mathit{nullable}(r_2)$$

$$\mathit{nullable}(r_1 \cdot r_2) \stackrel{\text{def}}{=} \mathit{nullable}(r_1) \wedge \mathit{nullable}(r_2)$$

$$\mathit{nullable}(r^*) \stackrel{\text{def}}{=} \mathit{true}$$

Zeroable

...whether a regular expression can match nothing:

$$\mathit{zeroable}(\emptyset) \stackrel{\text{def}}{=} \mathit{true}$$

$$\mathit{zeroable}(\epsilon) \stackrel{\text{def}}{=} \mathit{false}$$

$$\mathit{zeroable}(c) \stackrel{\text{def}}{=} \mathit{false}$$

$$\mathit{zeroable}(r_1 + r_2) \stackrel{\text{def}}{=} \mathit{zeroable}(r_1) \wedge \mathit{zeroable}(r_2)$$

$$\mathit{zeroable}(r_1 \cdot r_2) \stackrel{\text{def}}{=} \mathit{zeroable}(r_1) \vee \mathit{zeroable}(r_2)$$

$$\mathit{zeroable}(r^*) \stackrel{\text{def}}{=} \mathit{false}$$

Zeroable

...whether a regular expression can match nothing:

$$\mathit{zeroable}(\emptyset) \stackrel{\text{def}}{=} \mathit{true}$$

$$\mathit{zeroable}(\epsilon) \stackrel{\text{def}}{=} \mathit{false}$$

$$\mathit{zeroable}(c) \stackrel{\text{def}}{=} \mathit{false}$$

$$\mathit{zeroable}(r_1 + r_2) \stackrel{\text{def}}{=} \mathit{zeroable}(r_1) \wedge \mathit{zeroable}(r_2)$$

$$\mathit{zeroable}(r_1 \cdot r_2) \stackrel{\text{def}}{=} \mathit{zeroable}(r_1) \vee \mathit{zeroable}(r_2)$$

$$\mathit{zeroable}(r^*) \stackrel{\text{def}}{=} \mathit{false}$$

$$\mathit{zeroable}(r) \Leftrightarrow L(r) = \emptyset$$

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

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

- $A^* = \{\epsilon\} \cup A @ A^*$
- If $\epsilon \in A$, then

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

$$Der\ c\ (A @ B) = (Der\ c\ A) @ B$$

Grammars

$$\begin{aligned} E &\rightarrow F + (F \cdot " * " \cdot F) + (F \cdot "\" \cdot F) \\ F &\rightarrow T + (T \cdot " + " \cdot T) + (T \cdot " - " \cdot T) \\ T &\rightarrow \textit{num} + (" (" \cdot E \cdot ") ") \end{aligned}$$

E, *F* and *T* are non-terminals

E is start symbol

num, (,), + ...are terminals

$$(2*3)+(3+4)$$

$$E \rightarrow F + (F \cdot " * " \cdot F) + (F \cdot "\" \cdot F)$$

$$F \rightarrow T + (T \cdot "+" \cdot T) + (T \cdot "-" \cdot T)$$

$$T \rightarrow num + ("(" \cdot E \cdot ")")$$

$$(2 * 3) + (3 + 4)$$

