Automata and Formal Languages (4)

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

Last Week

Last week I showed you

- a tokenizer taking a list of regular expressions
- tokenization identifies lexeme in an input stream of characters (or string) and cathegorizes them into tokens

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.

```
"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"
```

Automata

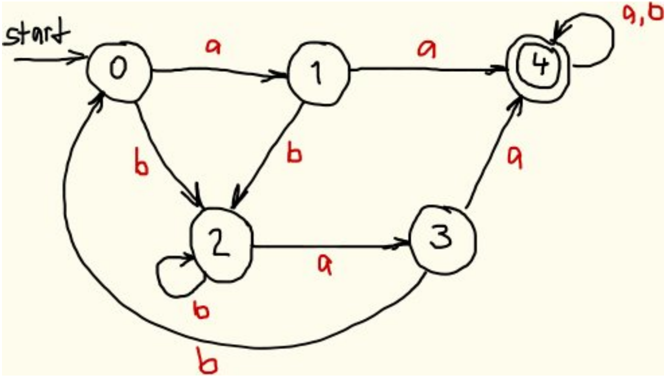
A deterministic finite automaton consists of:

- a finite 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 and a character as arguments and produces a new state

  this function might not always be defined everywhere

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


start
0
1
2
3
4
a
a
b
b
a
a
b
b
a,b

Accepting a String

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

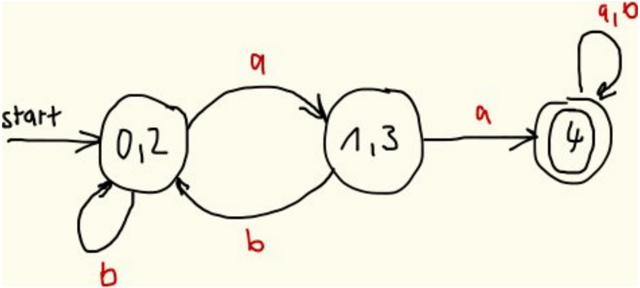
$$\hat{\delta}("", q) = q$$
$$\hat{\delta}(c :: s, q) = \hat{\delta}(s, \delta(c, q))$$

Accepting a String

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

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

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


start
0,2
1,3
4
a
a
b
b
a,b

Languages

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

Languages

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

not all languages are regular, e.g. $a^n b^n$.

- Assuming you have the alphabet $\{a, b, c\}$
- Give a regular expression that can recognise all strings that have at least one b.

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

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

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

- If "" $\notin A$, then
  $\text{Der}\, c\, (A\ @\ B) = (\text{Der}\, c\, A)\ @\ B$