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","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:

$$\texttt{"x - 3"}$$

OP:

"+", "−"

NUM:

(NONZERODIGIT · DIGIT*) + "0"

NUMBER:

NUM + ("−" · NUM)

Negation

Assume you have an alphabet consisting of the letters a, b and c only. Find a regular expression that matches all strings except ab, ac and cba.

Deterministic Finite Automata

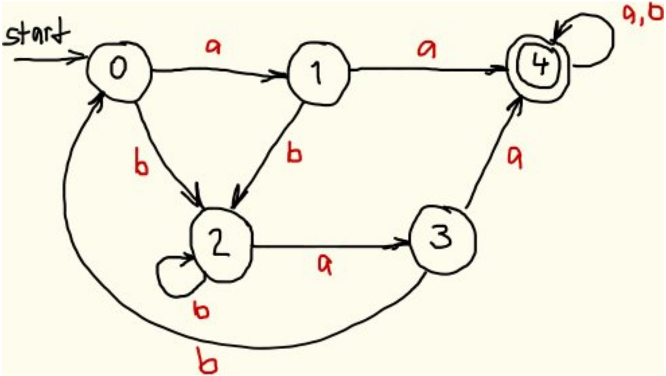
A deterministic finite automaton consists of:

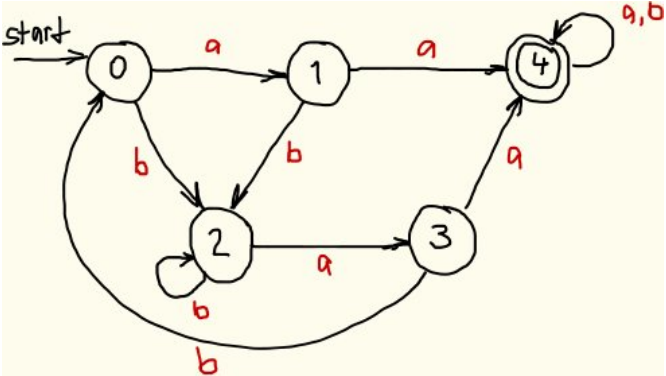
- a finite set of states
- one of these states is the start state
- some states are accepting states, and
- there is transition function

  which takes a state and a character as arguments and produces a new state

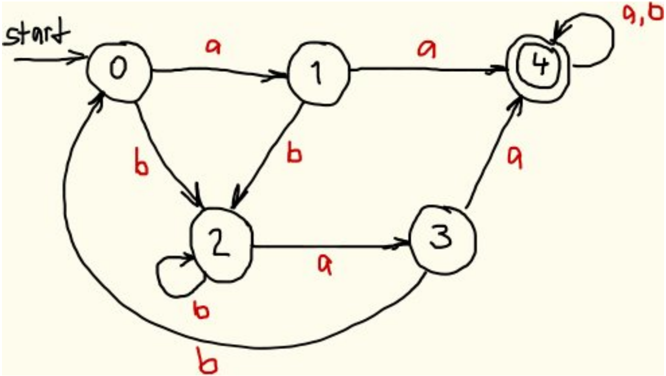
  this function might not always be defined everywhere

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


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



- start can be an accepting state
- it is possible that there is no accepting state
- all states might be accepting (but does not necessarily mean all strings are accepted)



for this automaton δ is the function

$$\begin{array}{lll} (q_0, a) \rightarrow q_1 & (q_1, a) \rightarrow q_4 & (q_4, a) \rightarrow q_4 \\ (q_0, b) \rightarrow q_2 & (q_1, b) \rightarrow q_2 & (q_4, b) \rightarrow q_4 \end{array} \quad \ldots$$

Accepting a String

Given

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

you can define

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

Accepting a String

Given

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

you can define

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

Whether a string s is accepted by A?

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

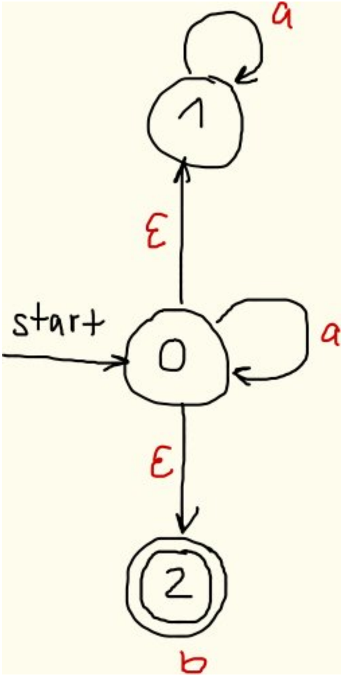
Non-Deterministic Finite Automata

A non-deterministic finite automaton consists again of:

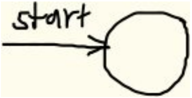
- a finite set of states
- one of these states is the start state
- some states are accepting states, and
- there is transition relation

$$(q_1, a) \rightarrow q_2$$
$$(q_1, a) \rightarrow q_3$$

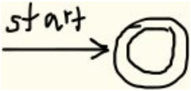
$$(q_1, \epsilon) \rightarrow q_2$$


a
1
ε
start
0
a
ε
2
b

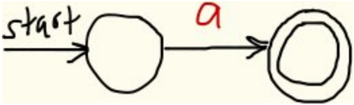
$\varnothing$



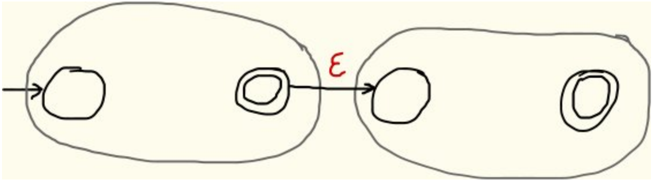
ϵ

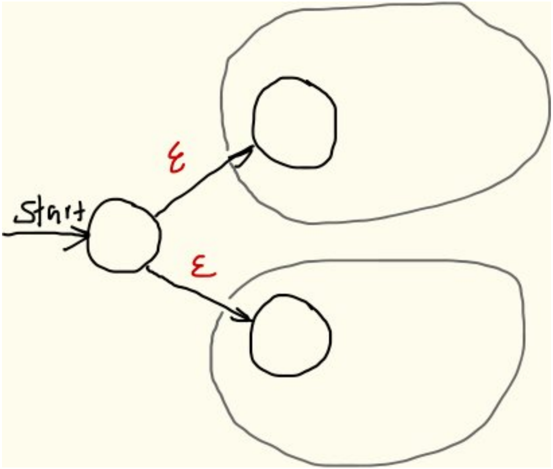


c

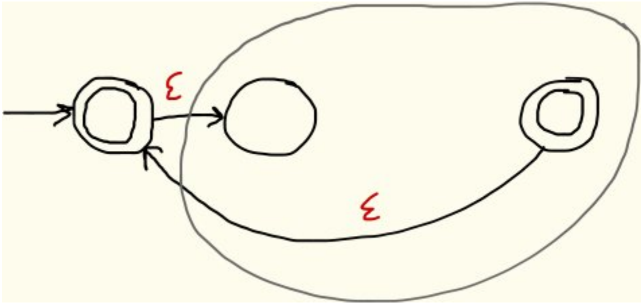


$\mathbf{r_1 \cdot r_2}$



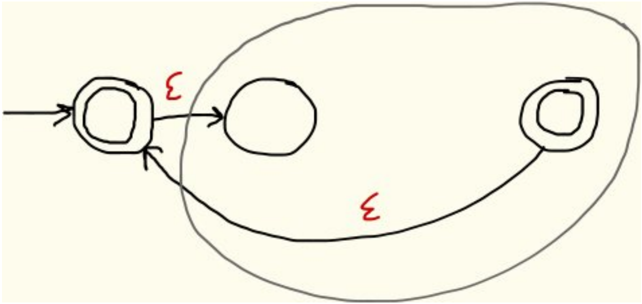


$r_1 + r_2$



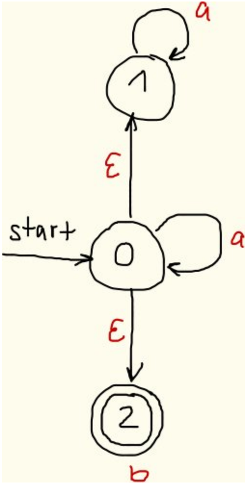
r^*

r^*



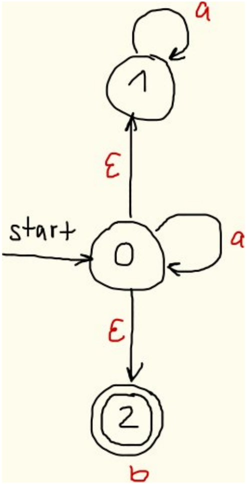
Why can't we just have an epsilon transition from the accepting states to the starting state?

Subset Construction



	a	b
$\varnothing$	$\varnothing$	$\varnothing$
$\{0\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\varnothing$
$\{2\}$	$\varnothing$	$\{2\}$
$\{0, 1\}$	$\{0, 1, 2\}$	$\{2\}$
$\{0, 2\}$	$\{0, 1, 2\}$	$\{2\}$
$\{1, 2\}$	$\{1\}$	$\{2\}$
$\{0, 1, 2\}$	$\{0, 1, 2\}$	$\{2\}$

Subset Construction



	a	b
$\varnothing$	$\varnothing$	$\varnothing$
$\{0\}$	$\{0,1,2\}$	$\{2\}$
$\{1\}$	$\{1\}$	$\varnothing$
$\{2\}$ *	$\varnothing$	$\{2\}$
$\{0,1\}$	$\{0,1,2\}$	$\{2\}$
$\{0,2\}$ *	$\{0,1,2\}$	$\{2\}$
$\{1,2\}$ *	$\{1\}$	$\{2\}$
s: $\{0,1,2\}$ *	$\{0,1,2\}$	$\{2\}$

Regular Languages

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

or equivalently

A language is regular iff there exists a deterministic finite automaton that recognises all its strings.

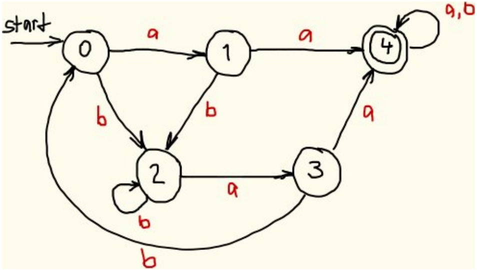
Regular Languages

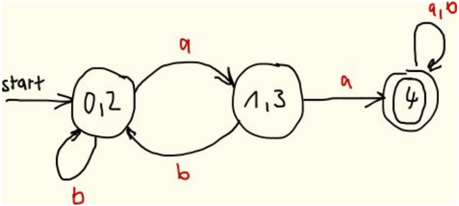
A language is regular iff there exists a regular expression that recognises all its strings.

or equivalently

A language is regular iff there exists a deterministic finite automaton that recognises all its strings.

Why is every finite set of strings a regular language?





minimal automaton

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.

Given the function

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

and the set

$$Rev\ A \stackrel{\text{def}}{=} \{s^{-1} \mid s \in A\}$$

prove whether

$$L(rev(r)) = Rev(L(r))$$

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

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

"I hate coding. I do not want to look at code."