Exercise 1

Consider a language with the following tokens and token classes:

ID ::= letter (letter|digit)*

LT ::= "<"

GT ::= ">"

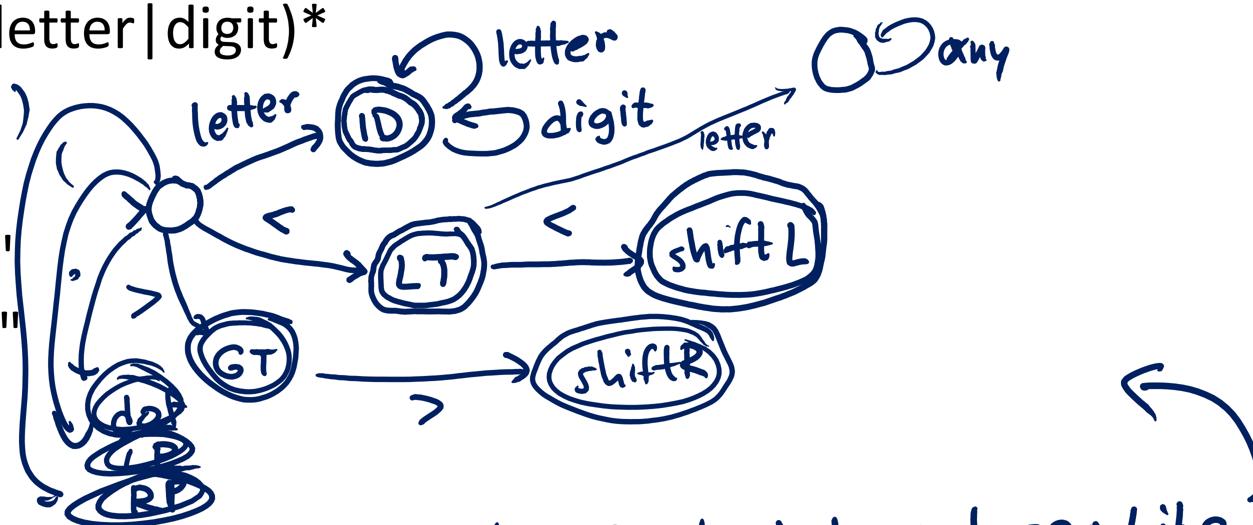
shiftL ::= "<<"

shiftR ::= ">>"

dot ::= "."

LP ::= "("

RP ::= ")"



a) Draw the automaton for lexical analyzer. Like so

b) Give a sequence of tokens for the following character sequence, applying the longest match rule:

shiftR

(List<List<Int>>)(myL).headhead

Note that the input sequence contains no space character

Exercise 2

$A = \{0, 1\}$

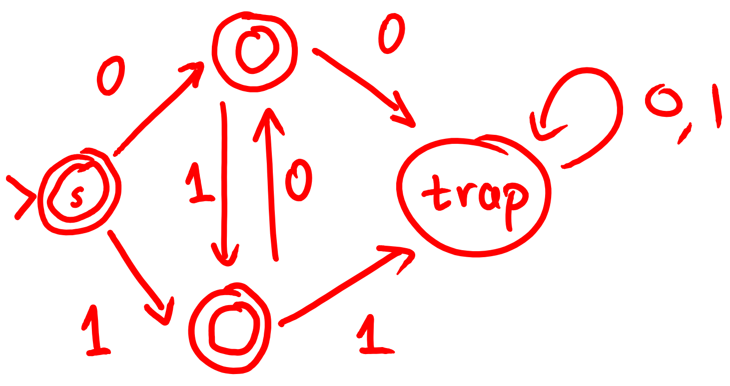
Find a regular expression that generates all alternating sequences of 0 and 1 with arbitrary length **(including lengths zero, one, two, ...)**. For example, the alternating sequences of length one are 0 and 1, length two are 01 and 10, length three are 010 and 101. Note that no two adjacent character can be the same in an alternating sequence.

$0(10)^* \mid 1(01)^* \mid (10)^* \mid (01)^*$

$(01)^* 0^? \mid (10)^* 1^?$

b)

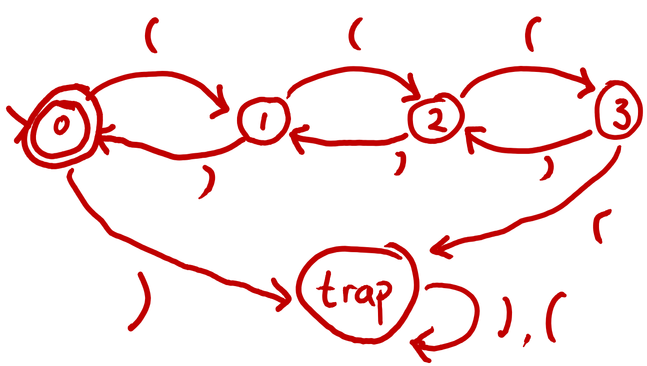
Automaton?


s
trap
0
0
0
1
1
1
0,1

Exercise 3

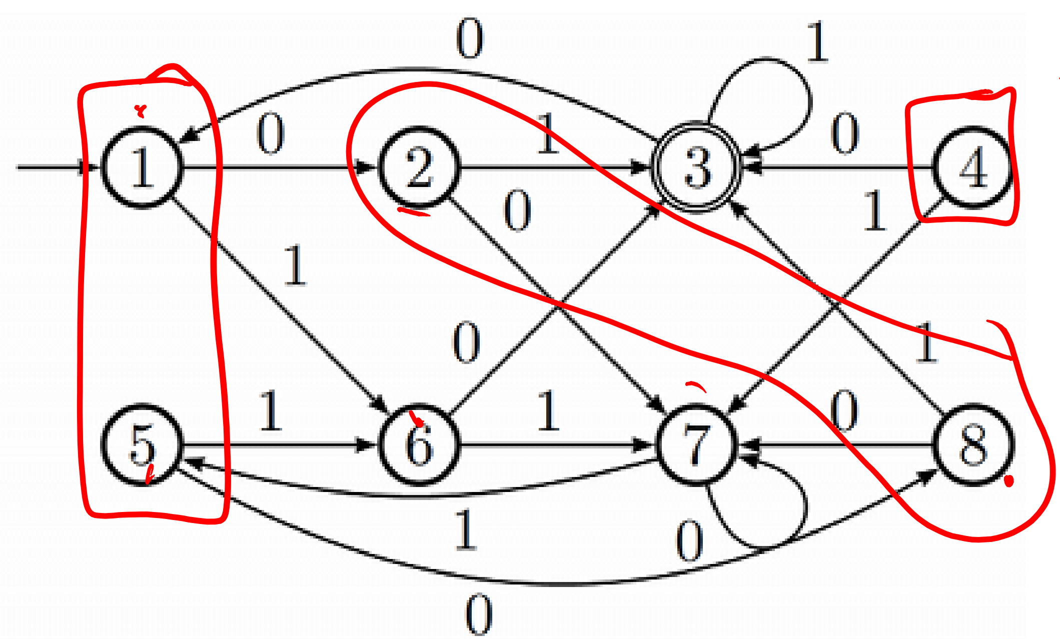
a) Describe any algorithm using a single unbounded integer counter that determines if a string consists of well-nested parentheses

b) Construct a DFA (deterministic finite-state automaton) for the language L of *well-nested* parenthesis of nesting depth at most 3. For example, ε, ()(), (()(())) and (()())()() should be in L,
but not (((()))) nor (()((())))), nor ())) .



Exercise 4

- Find two equivalent states in the automaton, and merge them to produce a smaller automaton that recognizes the same language. Repeat until there are no longer equivalent states.
- Recall that the general algorithm for minimizing finite automata works in reverse. First, find all pairs of inequivalent states. States X, Y are inequivalent if X is final and Y is not, or (by iteration) if and and X' and Y' are inequivalent. After this iteration ceases to find new pairs of inequivalent states, then X, Y are equivalent, if they are not inequivalent.



	1	2	3	4	5	6	7	8
1	×	≠	≠	≠	□	≠	≠	≠
2		×	≠	×	≠	≠	≠	□
3			×	≠	≠	≠	≠	≠
4				×	×	×	×	×
5					×	≠	≠	≠
6						×	≠	≠
7							×	≠
8								×

Exercise 5

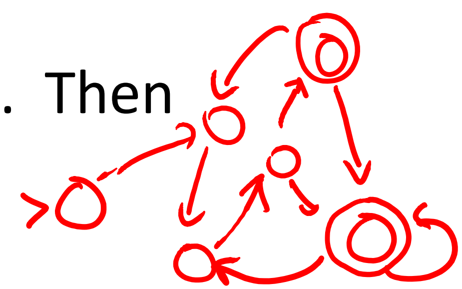
Let *tail* be a function that returns all the symbols of a string except the last one. For example

tail(mama)=mam

tail is undefined for an empty string. If $L_1 \subseteq A^*$, then TAIL(L_1) applies the function to all non-empty words in L_1, ignoring ε if it is in L_1: $\mathrm{TAIL}(L_1) = \{ v \in A^* \mid \exists\, a \in A.\ va \in L_1 \}$

TAIL({aba,aaaa,bb, ε}) = {ab,aaa,b}

L(r) denotes the language of a regular expression r. Then TAIL(L(abba|ba*|ab*)) = L(ba*|ab*|ε)



Tasks: $\{ q \mid \exists c.\ \delta(q,c) \in F \} = F'$

- Prove that if language L_1 is regular, then so is TAIL(L_1)
- Give an algorithm that, given a regular expression r for L_1, computes a regular expression rtail(r) for language TAIL(L_1)

Exercise 5 - solution

- You can first construct a regular expression or an automaton (whichever is convenient for you), and then convert one representation to the other using the standard algorithms.
- Alternatively, it is possible to define both regular expression and automata for tail(L) directly from the regular expression/automata of L

Approach I

a) First construct an automaton for tail(L)

If DFA for L is $(\Sigma, Q, q_0, \delta, F)$ then

DFA for tail(L) is $(\Sigma, Q, q_0, \delta, F')$

where

$$F' = \{ q \mid \exists c \in \Sigma. \delta(q, c) \in F \}$$

b) Convert the automaton for tail(L) to a regular expression

Exercise 5 - solution

- Approach II
  - First construct a regular expression for tail(L) using the following construction

$$rtail(r_1 \mid r_2) = rtail(r_1) \mid rtail(r_2)$$

$$rtail(r_1 r_2) = \begin{cases} r_1\, rtail(r_2) & , \neg nullable(r_2) \\ (r_1\, rtail(r_2) \mid rtail(r_1)) & nullable(r_2) \end{cases}$$

$$r_1 b$$

$$rtail(r^*) = r^* rtail(r)$$

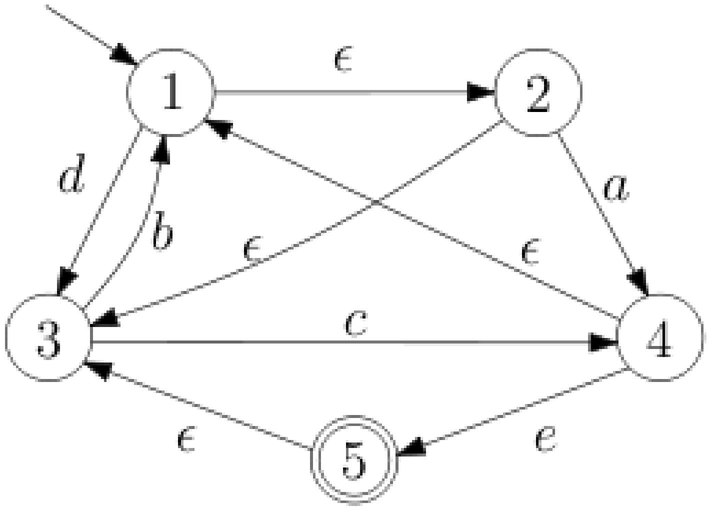
$$rtail((ab)^*) = (ab)^* a$$

  - Convert the regular expression to an automata

Exercise 6. Given NFA A, find first(L(A))

SKIP

- Compute the set of first symbols of words accepted by the following non-deterministic finite state machine with epsilon transitions:



- Describe an algorithm that solves this problem given a given NFA

More Questions

- Find automaton or regular expression for:
  - Any sequence of open and closed parentheses of even length?
  - as many digits before as after decimal point?
  - Sequence of balanced parentheses

    ((()) ()) - balanced

    ( )) (( ) - not balanced
  - Comment as a sequence of space,LF,TAB, and comments from // until LF
  - Nested comments like /* ... /* */ ... */

Automaton that Claims to Recognize $\{ a^n b^n \mid n >= 0 \}$

Make the automaton deterministic

Let the resulting DFA have K states, $|Q|=K$

Feed it a, aa, aaa, …. Let q_i be state after reading a^i

$q_0, q_1, q_2, \dots, q_K$

This sequence has length K+1 -> a state must repeat

$q_i = q_{i+p}$ $\quad\quad\quad\quad$ $p > 0$

Then the automaton should accept $a^{i+p}b^{i+p}$.

But then it must also accept

$$a^i b^{i+p}$$

because it is in state after reading a^i as after a^{i+p}.

So it does not accept the given language.

Limitations of Regular Languages

- Every automaton can be made deterministic
- Automaton has finite memory, cannot count
- Deterministic automaton from a given state behaves always the same
- If a string is too long, deterministic automaton will repeat its behavior

Pumping Lemma

If L is a regular language, then there exists a positive integer p (the pumping length) such that every string $s \in$ L for which $|s| \geq p$, can be partitioned into three pieces, $s = x\,y\,z$, such that

- $|y| > 0$
- $|xy| \leq p$
- $\forall i \geq 0.\ xy^iz \in$ L

Let's try again: { a^nb^n | n >= 0 }

Automata are Limited

Let us use **grammars**!