Exercise 1

Consider a language with the following tokens and token classes:

ID ident ::= letter (letter|digit)*

LT ::= "<"

GT ::= ">"

shiftL ::= "<<"

shiftR ::= ">>"

dot ::= "."

LP ::= "("

RP ::= ")"

LP ID LT ID LT ID $\underbrace{\text{shiftR}}_{\text{GTGT}}$ RP LP ID RP DOT ID

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

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

Note that the input sequence contains no space character

Exercise 2

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.

010101 101 1010

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

$0|1$ $x^?$

$1^?(01)^*0^?$

Exercise 3

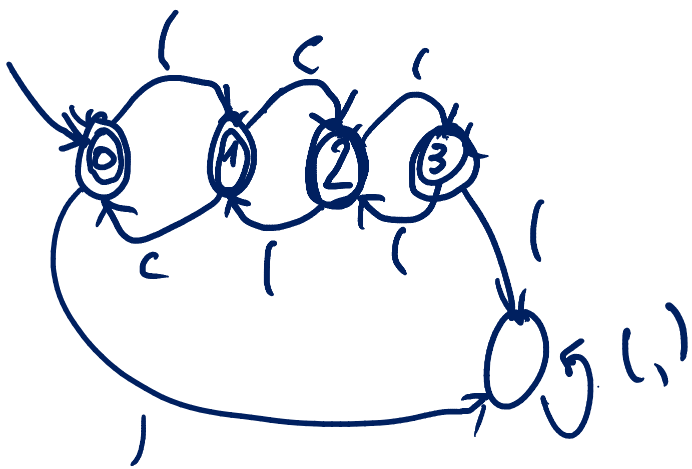
Construct a DFA for the language of well-nested parenthesis with a maximal nesting depth of 3. For example, ε, ()(), (()(())) and (()())()(), but not (((()))) nor (()(()(()))), nor ())).

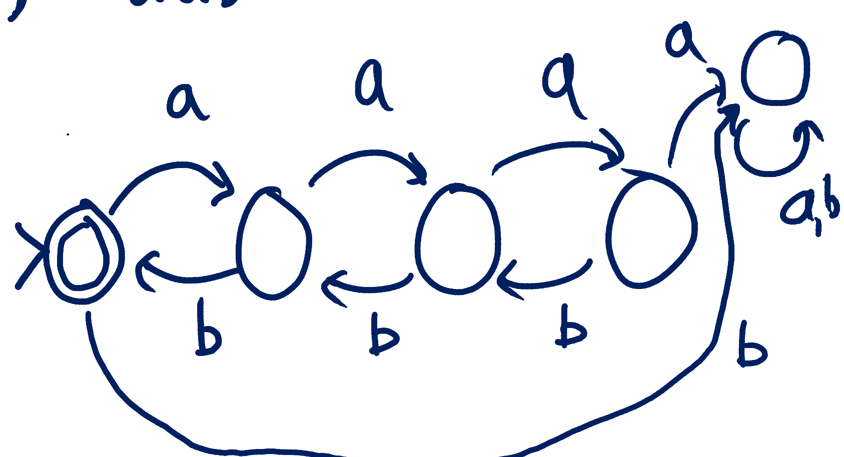
((( ) ( ) ( )))

a b

$\Sigma = \{ (,) \}$

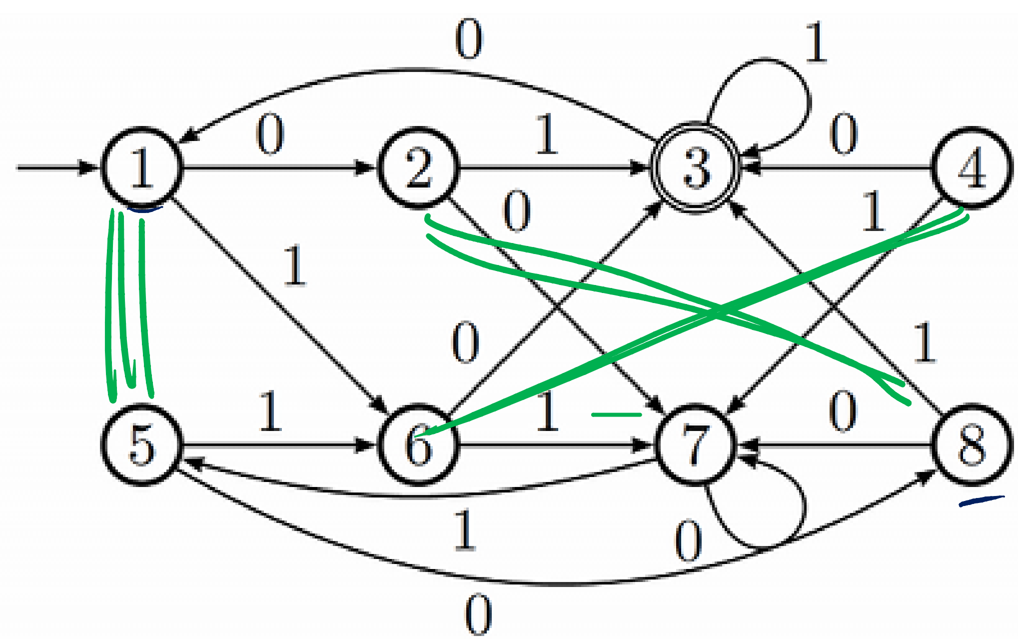
(() aab

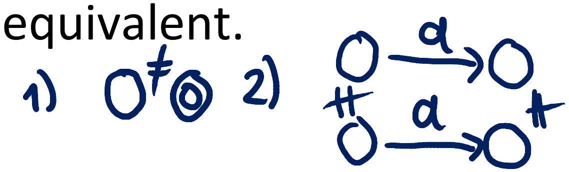




Exercise 4

- 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		X	X	X	O	X	X	X
2			X	X	X	X	X	□
3				X	X	X	X	X
4					X	□	X	X
5						X	X	X
6							X	X
7								X
8								

Exercise 5

Let *rtail* be a function that returns all the symbols of a string except the last one. For example, *rtail*(Lexer) = Lexe. *rtail* is undefined for an empty string. If R is a regular expression, then rtail(R) applies the function to all non-empty elements, and removes ε if it is in R. For example, rtail({aba,aaaa,bb, ε}) = {ab,aaa,b}

$$L(rtail(abba|ba^*|ab^*)) = L(ba^*|ab^*|\varepsilon)$$

- Prove that if R is regular, then so is rtail(R)
- Give an algorithm for computing the regular expression for rtail(R) if R is given by a regular expression

Exercise 6: Grammar Equivalence

Show that each string that can be derived by grammar G_1

$$B ::= \varepsilon \mid \mathbf{(}\ B\ \mathbf{)} \mid B\ B$$

can also be derived by grammar G_2

$$B ::= \varepsilon \mid \mathbf{(}\ B\ \mathbf{)}\ B$$

and vice versa. In other words, $L(G_1) = L(G_2)$

Remark: there is no algorithm to check for equivalence of *arbitrary* grammars. We must be clever.

Grammars

Ambiguous grammar: if some token sequence has multiple parse trees

(then it is has multiple abstract trees)

Two trees, each following the grammar, their leaves both give the same token sequence.

Exercise: Another Balanced Parenthesis Grammar

Show that the following balanced parentheses grammar is ambiguous (by finding two parse trees for some input sequence) and find unambiguous grammar for the same language.

$$B ::= \varepsilon \mid \mathbf{(}\ B\ \mathbf{)} \mid B\ B$$

Is this grammar ambiguous?

$$B ::= \varepsilon \mid \mathbf{(}\ B\ \mathbf{)}\ B$$

Dangling Else

The dangling-else problem happens when the conditional statements are parsed using the following grammar.

S ::= S **;** S

S ::= id **:=** E

S ::= **if** E **then** S

S ::= **if** E **then** S else S

if E_1 then
 if E_2 then
  $x := E_3$
else $y := E_4$

Give a sequence of tokens that has two parse trees.

Find an unambiguous grammar that accepts the same conditional statements and matches the else statement with the nearest unmatched if.