95+10

105

(1) a) DFA and NFA.

They both have starting states, sets of accept states, transition functions, input alphabet.

Difference: - DFA takes an input and start states to produce the next possible state.

- NFA takes an input and starting states to produce a set of possible next states.

Terminals are character of alphabets that appears in the string generated by the grammar.

Ex: boys / girls laugh / cry.

Nonterminals are the place holder for strings or language patterns of terminal symbols.

Nonterminals can be used to generate terminal symbols. Ex: <NOUN> <VERB>

b). Pumping Lemma is used as a proof for irregularity of a language => if language is regular, it satisfies pumping lemma.

. Alphabet is a finite set of fundamental units (letters or symbols)

. String: over an alphabet is a finite sequence of symbols from the alphabet (no spaces or commas between symbols).
. Language: is a set of strings over an alphabet.
. Sequence is a list of objects in some order.
. k-tuple: a finite sequence ~~that has k symbols from the alphabet~~ (k symbols (set of string) from alphabet).
. Ordered pair: ~~a pair of objects that its order is significant: $(a,b) \neq (b,a)$ unless $a=b$.~~
. Unordered pair: ~~a pair of objects that its order is not significant: (a,b) can be equal (b,a).~~
. ~~Domain~~ Ordered pair: a list of two elements.
. Unordered pair: a set with two members.
. Domain: The set of possible input to a function.
. Range: The set from which outputs of a function are drawn.

(2) CFG:
Context free grammar is a 4-tuple (V, Σ, R, S)
V: finite set called variable.
Σ: finite set of variable V, called terminal.
R: finite set of rules.
$S \in V$: state variable.

PDA:

Push down automation is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$

Q: set of states

Σ: input alphabet

Γ: ~~state~~ stack alphabet

$\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to P(Q \times \Gamma_\varepsilon)$ is the transition function.

$q_0 \in Q$: start state

$F \subseteq Q$: set of accept states.

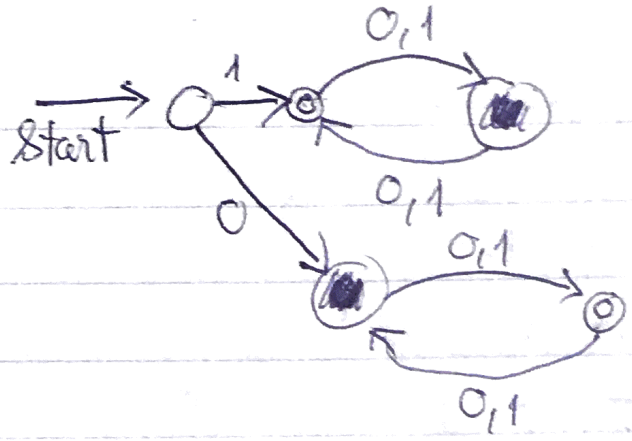
~~Regular~~

Regular expression:

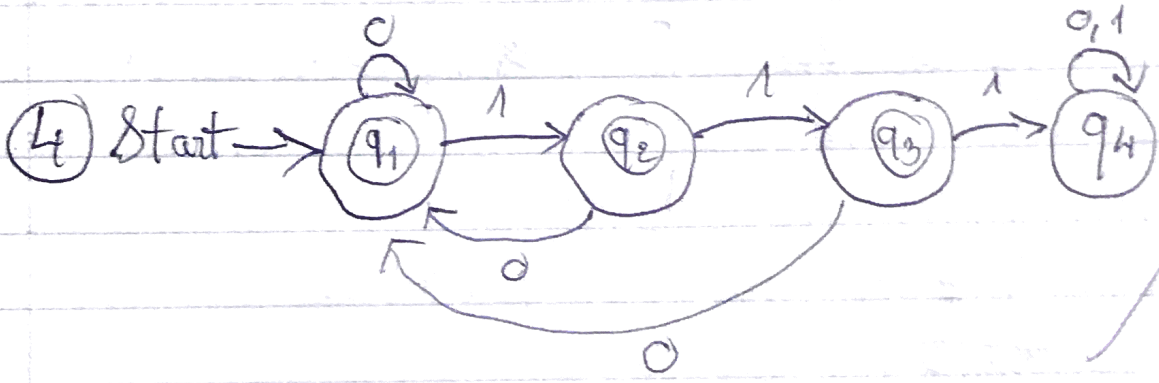
R is regular expression if R is

1. a for some a in the alphabet Σ.
2. ε
3. ϕ
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions.
6. (R_1^*) where R_1 is a regular expression.

(3)



(4)



$M = \{Q, \Sigma, \delta, q_0, F\}$

$Q = \{q_1, q_2, q_3, q_4\}$

$\Sigma = \{0, 1\}$

$\delta =$

	0	1
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_1	q_4
q_4	q_4	q_4

$q_0 = q_1$ (start state)

$F = (q_1, q_2, q_3)$ (final accept states)

$L = \{ w \mid w$ has at most 2 consecutive 1's $\}$

(5). PDA uses stack alphabet Γ, which allows PDA to recognize non-regular languages.

Limit: PDA has a transition function that takes an input and a state or an empty string; DFA needs to have input and state to produce next state. (cannot take empty string)

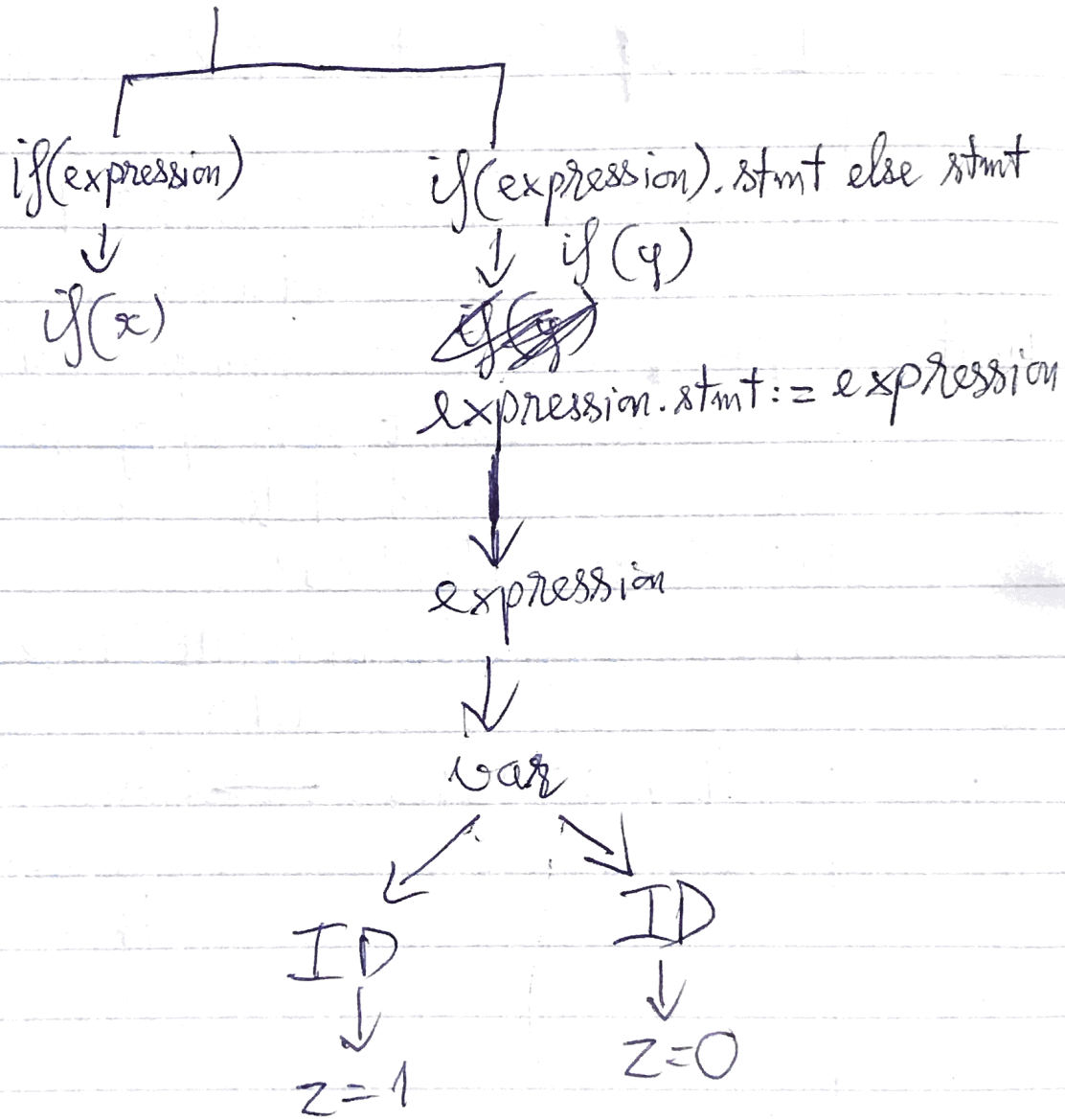
<PARENTHESIS>

(|)

<PARENTHESIS> → <PARENTHESIS>
<PARENTHESIS>
|(<PARENTHESIS>|
|<PARENTHESIS>|

(6) ~~comp~~
~~(6)~~ compounds

b)



a) compound-stmt c-s
selection-stmt s-s
iteration-stmt ~~i-is~~ i-s
return-stmt r-s
break-stmt b-s
expression-stmt e-s -> e
:= expression

selection-stmt ::= if (expression) stmt / s-s	if (e) s
if (expression) stmt else stmt	if (e) s else s
expression-stmt ::= expression;	e -> var = e
var += expression	var (+) = e
var = expression	var = e
simple-expression	e (s)
var ::= ID	var = ID
ID [expression]	ID (e)

10