EXAMINATION BLUE BOOK

Name Dang Nhi Ngo

Subject Foundation of Computer Science

Instructor

Section Class

Date 10/18/2018 Book No.

95+5

15

1/ a/ Differentiate between

– DFA , NFA

+ DFA has exactly 1 exiting transition arrow for each symbol of the alphabet

NFA has zero, more or many exiting transition arrow for eac symbol of the alphabet

+ DFA : labels on the transition arrow are symbols from the alphabet

NFA : labels on the transition arrow are symbols from the alphabet or ε (empty string)

– Terminal VS Non-terminal (variable)

Terminal is the string including variables and other symbols It is similar as input alphabet and is represented by lowercase letters, numbers or special symbols. Terminal is generated by the grammar

While, non-terminals are the placeholders for patterns of terminal symbols that can be generated by the non-termin symbols b/ Define

- Pumpilng lemma for Regular languages : Technique for proving nonregularity stems from a theory about regular languages
- Alphabet : A finite, nonempty set of objects called symbols
- String : A finite list of symbols from an alphabet
- Language : A set of strings
- Sequence is the list of objects in some order
- k- tuple] : A list of k objects
- Ordered pair : A list of 2 elements
- Unordered pair : A set with 2 members
- Domain is a set of possible inputs of a function
- Range is a set of outputs of a function

2/ Formal definition for

- CFG (Context Free Grammar) is a 4-tuple of

  (V, Σ, R, S) where

  V: finite set called variables

  Σ. finite set, disjoint from V called terminals

  R: a finite set of rules, with each rule being a variable and a string of variables and symbols

  S: start variable ($S \in V$)

- PDA (Pushdown automata):

  It is like NFA but have an extra component called stack

  6-tuple of (Q, Σ, Γ, δ, q_0, F), where

  Q: finite set called states

  Σ: finite set called input alphabet

  Γ: finite set called stack alphabet

  δ : $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ (transition function)

  q_0 : start state

  F : finite set of accept states

PDA has stack which is used to provide additional memory beyond the finite amount available in the control. It also allows PDA to recognize non-regular language

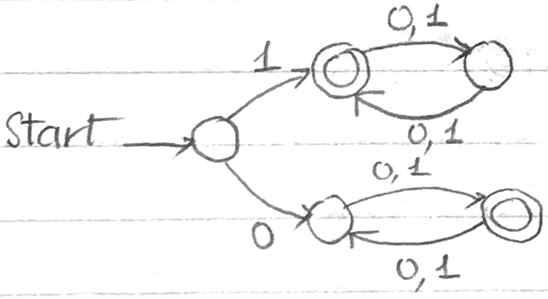
– Regular Expression :

R is a regular expression if R is

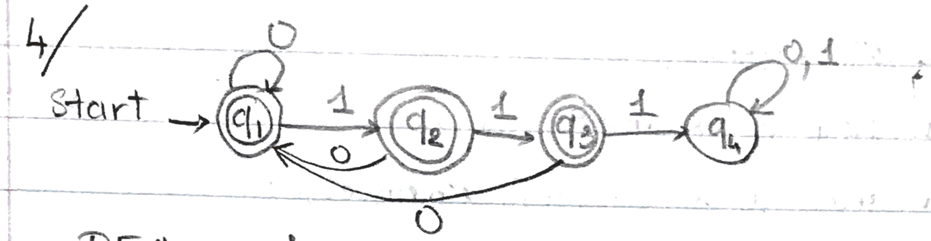
- + a for some a is in the alphabet Σ
- + ∅
- + ε
- + $(R_1 \cup R_2)$ with R_1 and R_2 are regular expressions
- + $(R_1 \circ R_2)$ with R_1 and R_2 are regular expressions
- + $(R_1{*})$ with R_1 is regular expression

3/

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



4/



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

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

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

$q_0 = \{q_1\}$

$F = \{q_1, q_2, q_3\}$

Q is a finite set called states

Σ is a finite set called alphabet

q_0 : start state is q_1

F is a finite set of accept states

Transition function:

	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

This machine accepts regular language

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

5/ * PDA differs from a DFA

- PDA has stack which provides additional memory beyond the finite amount available in the control. Stack allows PDA to recognize non-regular language
- PDA uses the top of stack to decide which transition to take
- PDA can manipulate the stack as part of performing a transition
- PDA has pushing (add a symbol to the stack) and popping (remove a symbol from a stack)
- DFA does not have the capability to store long sequence of input alphabets, while PDA does thanks to the stack component
- DFA : Input alphabets are accepted by reaching 'final stat

  PDA : Input alphabets are accepted by reaching 'final state' and 'Empty stack'
- DFA is constructed for regular language

  PDA is constructed for context free grammar

* The limitation of a DFA that is addressed by a PDA :

DFA needs to have input and state to produce next state, while PDA can accept input alphabet and empty string ε

Also, PDA has stack component that allows to store long sequence of input alphabets and recognizes non-regular language which DFA can not.

* The grammar for generating matching parenthesis

<PARENTHIESIS>

( )

<PARENTHIESIS> → <PARENTHIESIS>

<PARENTHIESIS>

| <PARENTHIESIS> |

| <PARENTHIESIS> |.

6/

compound - stmt	c - s
selection - stmt	s - s
iteration - stmt	i - s
return - stmt	r - s
break - stmt	b - s

expression - stmt ::= expression   e-s → e

selection - stmt ::= if (expression) statement

s → s if (e) s

if (e) s else s

expression - stmt ::= expression

e ⟶ var = e

var (+) = e

var = e

e(s)

var = ID

ID(e)

۱ [illegible]

۲ [illegible]

۳ [illegible]

۴ [illegible]

۵ [illegible]