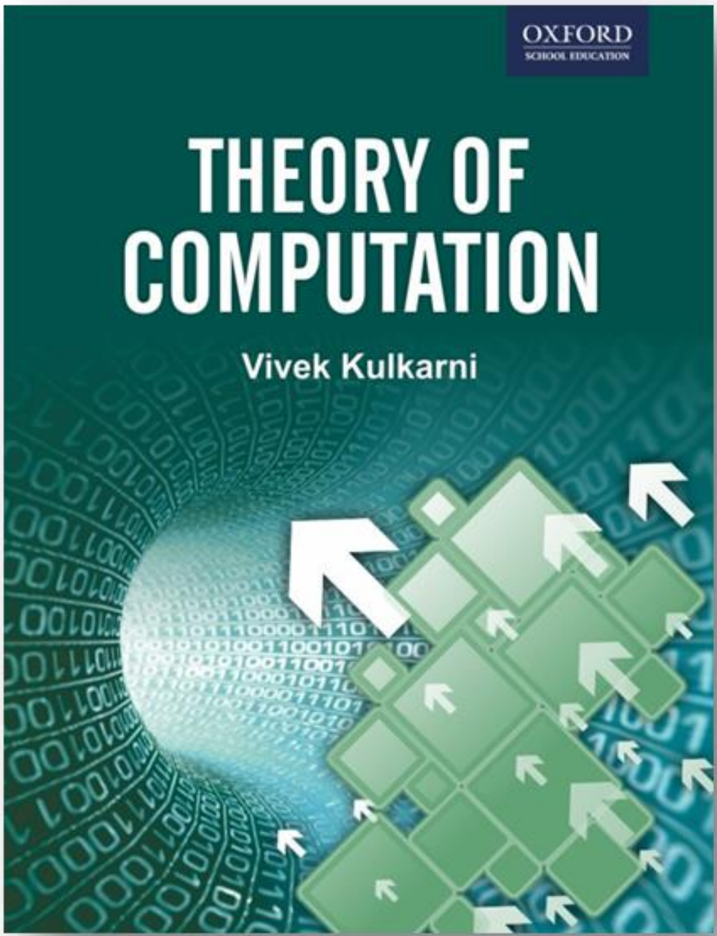


Author: Vivek Kulkarni
(vivek_kulkarni@yahoo.com)

Solution for

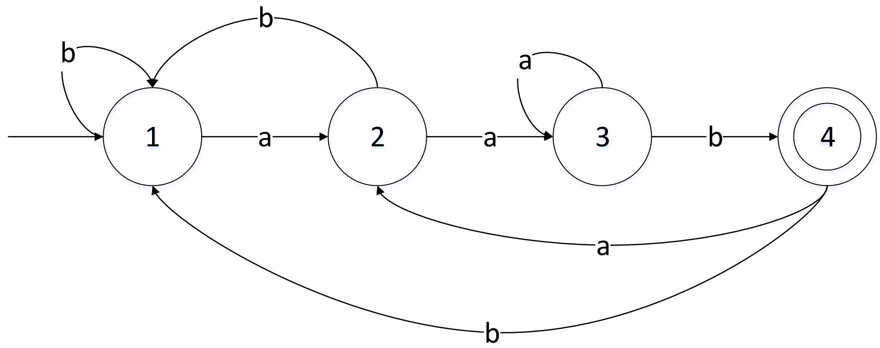
Model Question Paper 3

(From Appendix B)

Q.1 a) NFA that accepts any positive number of occurrences of various strings from the following language L given as,

L = {x | x is made up of {a, b} and x ends with "aab"}

	a	b
1	2	1
2	3	1
3	3	4
4	2	1



Q.1 b) Refer to the example 2.30 from the book.

Q.1 c) Let us first find out the $\in$-closure of each of the states.

$\in$-closure (A) = {A, B, C}

$\in$-closure (B) = {B}

$\in$-closure (C) = {C}

$\in$-closure (D) = {D}

D is the only final state even for the resultant NFA without $\in$-moves as only *D* is at zero distance from itself.

The state transition function, δ', for the required NFA without $\in$-moves is given as:

$$\delta' (q, a) = \in\text{-closure} (\delta (\hat{\delta} (q, \in), a))$$

where, $\hat{\delta} (q, \in) = \in\text{-closure} (q)$

For example,

δ' (a, 0) $=$ $\in$-closure (δ ($\hat{\delta}$ (a, $\in$), 0))

$=$ $\in$-closure (δ ({A, B, C}, 0)

$=$ $\in$-closure ({C, D})

$=$ {C, D}

δ' (a, 1) $=$ $\in$-closure (δ ($\hat{\delta}$ (a, $\in$), 1))

$=$ $\in$-closure (δ ({A, B, C}, 1)

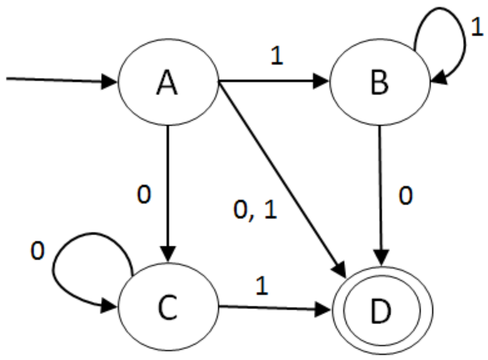
$=$ $\in$-closure ({B, D})

$=$ {B, D}

We can complete the STF, δ' for the equivalent NFA without $\in$-moves as below,

Q\∑	0	1
A	{C, D}	{B, D}
B	{D}	{B}
C	{C}	{D}
* D	ϕ	ϕ

The transition diagram for the NFA without $\in$-moves is,



Q.2 **a)** (i) aaaaaa

(ii) abbbbbbb

(iii) a

Q.2 **b)** Length of every string in *L* is a prime number.

Step 1: Let us assume that the language *L* is a regular language. Let *n* be the constant of the pumping lemma.

Step 2: Let us choose a sufficiently large string z such that $z = 0^l$, for some large $l > 0$; the length of z is given by: $|z| = l \geq n$.

Since we assumed that L is a regular language and from the language definition it is an infinite language, we can now apply pumping lemma.. This means that we should be able to write z as: $z = uvw$.

Step 3: As per pumping lemma, every string 'uv^iw', for all $i \geq 0$, is in L. Likewise, $|v| \geq 1$, which means that v cannot be empty and must contain one or more symbols.

Let us consider the case when v contains a single symbol:

In this case, $z = uvw = 0^l$, which means that the number of 0's in z is a prime number. As per pumping lemma, we would expect 'uv^2w' also to be a member of L; however, this cannot be possible, as v contains only a single symbol, and adding one to the prime number length would not always yield perfect prime length. Thus, pumping v would yield strings with non-prime lengths. Thus, 'uv^2w' is not a member of L. This contradicts our assumption that L is regular.

Let us now consider the case when v contains perfect prime number of 0's. A sample v could be written as: '000' (three 0's), or '00000' (five 0's), and so on. When we try to pump v multiple times, such as, for example, v^2 = 000000 (six 0's), or v^2 = 0000000000 (10 0's), and so on, we find that the length does not remain a perfect prime, and we get a string which is against the language definition, which is '0^i'. Thus, we can say that 'uv^2w' is not a member of L. This contradicts our assumption that L is regular.

Similarly, if we consider that v contains any number of 0's, then on pumping it we will get into a situation where the string has non-prime length, which is against the language definition. For example, if v contains 2 zeros and if we pump it say 2 times, we will get the string "0000" which does not have a perfect prime length.

Hence, the language $L = \{0^n \mid n \textit{ is a prime number}\}$ is non-regular.

Q.2 **c)** Refer to the example 3.27 from the book.

Q.3 **a)** Refer to the example 5.6 from the book.

Q.3 **b)** Refer to the section 5.17.

Q.3 c) Refer to the example 5.52 from the book.

Q.3 d) Refer to the example 5.43 from the book.

Q.3 e) The language, $L = \{a^n b^m \mid n = m\}$ is context-free. The CFG for the same is,

$$S \rightarrow a\,S\,b \mid \in$$

Q.4 a) Refer to the example 6.11 from the book.

Q.4 b)

(i) Actually CFLs are closed under intersection.

(ii) Refer to the section 6.8, theorem 6.3.

Q.4 c) Refer to the section 6.6.

Q.5 a) Refer to the example 4.13 from the book.

Q.5 b) Refer to the section 4.9.

Q.5 c) Refer to the section 4.15.

Q.6 a) Let us first construct the LR (1) sets of items. We need to augment the grammar to begin with.

Augmented grammar:

0: $D' \rightarrow D$

1: $D \rightarrow L : T$

2: $L \rightarrow L, id$

3: $L \rightarrow id$

4: $T \rightarrow int$

5: $T \rightarrow real$

LR (1) sets of items are:

I0:

$D' \rightarrow \cdot D \qquad , \$$

$D \rightarrow \cdot L : T \qquad , \$$

$L \rightarrow \cdot L , id \qquad , : / ,$

$L \rightarrow \cdot id \qquad , : / ,$

I1:

$D' \rightarrow D \cdot \qquad , \$$

I2:

$D \rightarrow L \cdot : T \qquad , \$$

$L \rightarrow L \cdot , id \qquad , : / ,$

I3:

$L \rightarrow id \cdot \qquad , : / ,$

I4:

$D \rightarrow L : \cdot T \qquad , \$$

$T \rightarrow \cdot int \qquad , \$$

$T \rightarrow \cdot real \qquad , \$$

I5:

$L \rightarrow L , \cdot id \qquad , : / ,$

I6:

$D \rightarrow L : T \cdot \qquad , \$$

I7:

$T \rightarrow int \cdot \qquad , \$$

I8:

$T \rightarrow real \cdot \qquad , \$$

I9:

$L \rightarrow L , id \cdot \qquad , : / ,$

The canonical-LR parsing table can be shown as below. As it cannot be reduced further it is the LALR parsing table as well.

State	ACTION						*GOTO*		
	:	,	id	real	int	$	D	L	T
0			s3				1	2	
1						Accept			
2	s4	s5							
3	r3	r3							
4				s8	s7				6
5			s9						
6						r1			
7						r4			
8						r5			
9	r2	r2							

Let us simulate the working of the parser for the input string '*a*, *b*, *c* : int'. Here *a*, *b*, and *c* are considered as *id*.

Stack	**Input**	**Parser Action**
0	*a*, *b*, *c* : int $	Shift '0' initially
0 *a* 3	, *b*, *c* : int $	*ACTION* (0, *id*) = *s*3
0 *L* 2	, *b*, *c* : int $	*ACTION* (3, ,) = *r3* *GOTO* (0, *L*) = 2
0 *L* 2 , 5	*b*, *c* : int $	*ACTION* (2, ,) = *s5*
0 *L* 2 , 5 b 9	, *c* : int $	*ACTION* (5, id) = *s9*
0 *L* 2	, *c* : int $	*ACTION* (9, ,) = *r2* *GOTO* (0, *L*) = 2
0 *L* 2 , 5	*c* : int $	*ACTION* (2, ,) = *s5*
0 *L* 2 , 5 c 9	: int $	*ACTION* (5, id) = *s9*
0 *L* 2	: int $	*ACTION* (9, :) = *r2* *GOTO* (0, *L*) = 2
0 *L* 2 : 4	int $	*ACTION* (2, :) = *s*4
0 *L* 2 : 4 int 7	$	*ACTION* (4, int) = *s7*
0 *L* 2 : 4 T 6	$	*ACTION* (7, $) = *r*4 *GOTO* (4, *T*) = 6
0 *D* 1	$	*ACTION* (6, $) = *r*1

Stack	**Input**	**Parser Action**
		GOTO (0, *D*) = 1
0 *D* 1	$	'ACCEPT', because *ACTION* (1, $) = 'accept'

Q.6 b) Refer to the example 11.12 from the book.

Q.6 c) (i) Gödel numbering: Refer to the section 9.3.

(ii) Diagonalization language: Refer to the section 9.4.1.

Q.6 d) Refer to the section 9.2.1, theorem 9.1.

