Solutions: Homework Set # 5

Problem 1

(a) An FSM is uniquely decodable if we can reconstruct the input sequence using the initial state of the FSM and its output, i.e., if we start from a state and feed two different inputs, we get different outputs.

Similarly, an FSM is called information lossless if having the output, the initial and final states, one can uniquely determine the input sequence.

It is clear that being IL is a necessary but not sufficient condition for being UD.

We claim that FSM1 is UD and therefore it is also IL. Assume that it is not. Therefore there exist two different sequences x^n and z^m and an initial state w which yield in the same output, y^p. Let t be the first position where x^n and z^m are different, i.e., $x^{t-1} = z^{t-1}$ and $x_t \neq z_t$. Without loss of generality one can assume $x_t = a$ and $z_t = b$. Assume that Since $x^{t-1} = z^{t-1}$, we will be in the same state when x_t or z_t are fed to the FSM, namely w_t, and $q-1$ of the output symbols are already produced. Thus, y_q would be the first next output symbol. However it is easy to see that the first output symbol produced by the FSM is different for different inputs:

state	input	first bit of the output
s	a	0
s	b	1
x	a	1
x	b	0
y	a	1
y	b	0

FSM2 is information lossless but not uniquely decodable. One can check that starting from state s, both the two inputs "ba" and "aba" result in output "111". However, if we know the final state, two different input cannot yield in the same output. It is clear that the final state cannot be s (unless for the null input sequence). If the final state is x, it is clear that we have always been in the left branch of the FSM, otherwise we have always been in the right branch. In both cases it is easy to show that the FSM is information lossless.

(b) The output sequence is: "10101010111"

(c) The sequence can be parsed into 6 distinct words as $bb/ab/aa/ba/b/a$. This is in fact the maximum, because the minimum length of a sequence can be parsed into 7 distinct words is $n_7 = 2^1 \cdot 1 + 2^2 \cdot 2 + 1 \cdot 3 = 13 > 10$.

(d) The following table shows how the LZ algorithm works.

dictionary	codewords	new w.	codeword	output
$\{a, b\}$	$\{0, 1\}$	b	1	1
$\{a, ba, bb\}$	$\{00, 01, 10\}$	ba	01	101
$\{a, baa, bab, bb\}$	$\{00, 01, 10, 11\}$	baa	01	10101
$\{a, baaa, baab, bab, bb\}$	$\{000, 001, 010, 011, 100\}$	bab	011	10101011
$\{a, baaa, baab, baba, babb, bb\}$	$\{000, 001, 010, 011, 100, 101\}$	a	000	10101011000

So, the Lempel-Ziv algorithm encodes this sequence to "10101011000" whose length is 11.

(e) As can be seen in the above table, the LZ algorithm parses the sequence into 5 words, $\{b, ba, baa, bab, a\}$.

Problem 2

(a) The initial dictionary is $X_0 = \{a, b, c, d\}$,

X_0	a	b	c	d
dic	0	1	2	3
bits	00	01	10	11

We can parse the sequence into parses we haven't seen until now. So, *abadcdadd* will be parsed into a, b, ad, c, d, add. At each iteration, we output the binary representation of the parsed substrings and substitute it in the dictionary with all its single letter extensions.

Step 1:

The output will be the representation of a: "00"

X_0	aa	ab	ac	ad	b	c	d
dic	0	1	2	3	4	5	6
bits	000	001	010	011	100	101	110

Step 2: The output will be the representation of b: "100"

X_0	aa	ab	ac	ad	ba	bb	bc	bd	c	d
dic	0	1	2	3	4	5	6	7	8	9
bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Step 3: The output will be the representation of ad: "0011"

X_0	aa	ab	ac	ada	adb	adc	add	ba	bb	bc	bd	c	d
dic	0	1	2	3	4	5	6	7	8	9	10	11	12
bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

Step 4: The output will be the representation of c: "1011"

X_0	aa	ab	ac	ada	adb	adc	add	ba	bb	bc	bd	ca
dic	0	1	2	3	4	5	6	7	8	9	10	11
bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011

X_0	cb	cc	cd	d
dic	12	13	14	15
bits	1100	1101	1110	1111

Step 5: The output will be the representation of d: "1111"

X_0	aa	ab	ac	ada	adb	adc	add	ba	bb	bc	bd
dic	0	1	2	3	4	5	6	7	8	9	10
bits	00000	00001	00010	00011	00100	00101	00110	00111	01000	01001	01010

X_0	ca	cb	cc	cd	da	db	dc	dd
dic	11	12	13	14	15	16	17	18
bits	01011	01100	01101	01110	01111	10000	10001	10010

Step 6: The output will be the representation of *add*: "00110"

So the final string will be: 0010000111011111100110.

(b) Using the fact that the dictionary grows by 3 elements with each parsing, we can parse the given sequence as: $00, 100, 0001, 0001, 1111$

	00	100	0001	0001	1111
$\|X\|$	4	7	10	13	16

Step 1:

"00" $= a$

X_0	aa	ab	ac	ad	b	c	d
dic	0	1	2	3	4	5	6
bits	000	001	010	011	100	101	110

Step 2: "100" $= b$

X_0	aa	ab	ac	ad	ba	bb	bc	bd	c	d
dic	0	1	2	3	4	5	6	7	8	9
bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Step 3: "0001" $= ab$

X_0	aa	aba	abb	abc	abd	ac	ad	ba	bb	bc	bd	c	d
dic	0	1	2	3	4	5	6	7	8	9	10	11	12
bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

Step 4: "0001" $= aba$

X_0	aa	abaa	abab	abac	abad	abb	abc	abd	ac	ad	ba	bb
dic	0	1	2	3	4	5	6	7	8	9	10	11
bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011

X_0	bc	bd	c	d
dic	12	13	14	15
bits	1100	1101	1110	1111

Step 5: "1111" $= d$

So, the Decoded sequence will be : *abababad*

(c) We can parse the given sequence as:
$a, b, abab, ab, bb, a, abb, aab$
$(0, a), (0, b), (2, 6), (1, 2), (4, 1), (5, 3), (4, 3)$

If we label $a = 0$ and $b = 1$, the output sequence is:

0000⁝0001⁝010110⁝001010⁝100001⁝110011⁝100011

(d)

000, 0	000, 1	010, 010	011, 110	101, 100
$(0, a)$	$(0, b)$	$(2, 2)$	$(3, 5)$	$(5, 4)$

The sequence will be:
$a, b, ab, babba, babb.$

Problem 3

Encoding:

The dictionary gets initialized to $[A, B, C, D]$, $[.c.]$ is empty, and $K = A$. The if condition enters the block Yes, and $[.c.] = A$. The next symbol is $K = B$ and as $[.c.]K = AB$ is not in the dictionary, the if condition enters No block , the algorithm outputs "$00''$" and AB is added to the dictionary and $[.c.] = B$. So now the dictionary is $[A, B, C, D, AB]$. The next symbol is A and the algorithm works exactly as described; until for the last symbol, when $K = B$, $[.c.]K = AB$ is in the dictionary, the output is "$0100''$", the dictionary gets updated, $[.c.] = AB$, and finally there is no next character and the encoding stops: Encoded stream: 000010000101000100.

Decoding: The dictionary is initialized with $[A, B, C, D]$ and the code to read is "$00''$". So the algorithm outputs A at the beginning and $OLDCODE := A$. the next code is "001". (you always read $\log(|D| + 1)$ bits except for the very first code you read, have it was "$00''$") the if condition is true, and thus the output is B. before going out of the Yes block,$[...] = A$, $K = B$, $[...]K = AB$ is added to dictionary, the dictionary gets updated to $[A, B, C, D, AB]$, and finally $OLDCODE = 01$. The next code is "$000''$" and decoding continues as described. One thing to note is that the order of the elements of the dictionary never changes and thus "$01''$, "$001''$, "$0001''$, $\cdots$ are always the second element of the dictionary. In the last step of decoding, the dictionary is already $[A, B, C, D, AB, BA, AC, CA]$ and thus the code to read is 0100 (as $\lceil \log_2(8 + 1) \rceil = 4$). The if condition is true, the algorithm outputs AB, $[...] = AB$, $K = A$, ABA is added to the dictionary, $OLDCODE = 0100$, and then there is no more code and decoding stops.

Problem 4

(a) The stationary distribution is $\pi = [p_0, p_1]$, such that $\pi P = \pi$, where

$$P = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

Thus, $\pi = [\frac{3}{7}, \frac{4}{7}]$

(b) The form of the sequence of states from state 0 returning to state 0 for the first time would be

$0\overbrace{11...1}^{l}0$ for $l = 0, 1, ...$

And each has a returning time of $l + 1$. So on average we have

$$\mathbb{E}(\text{returning time to } 0) = \sum p(X_1...X_{l+2} = 011..10|X_1 = 0).(l + 1)$$

$$= p(X_1X_2 = 00|X_1 = 0).1 + p(X_1X_2X_3 = 101|X_1 = 0).2 + \sum p(X_1X_2...X_{l+2} = 0\overbrace{11..1}^{l}0).(l + 1)$$

$$= p_{0,0} + p_{0,1}p_{1,0}.2 + \sum_{l=2}^{\infty} (l + 1)p_{01}(p_{11})^{l-1}p_{10}$$

$$= \tfrac{1}{3} + \tfrac{2}{3} + \tfrac{2}{3} \textstyle\sum_{l=2}^{\infty} (l + 1)(\frac{1}{2})^2$$

$$= \tfrac{1}{3} + \tfrac{2}{3} \underbrace{\sum_{l=1}^{\infty} (\frac{1}{2})^l}_{=1} + \tfrac{2}{3} \underbrace{\sum_{l=1}^{\infty} l(\frac{1}{2})^l}_{=2(*)}$$

$= \frac{7}{3} + \frac{1}{p_0}$

(*): $\sum_{l=1}^{\infty} l(\frac{1}{2})^l = \frac{1}{2} + 2 \times \frac{1}{4} + 3 \times \frac{1}{8} + ...$

$= \frac{1}{2}+$

$\frac{1}{4} + \frac{1}{4}+$

$\frac{1}{8} + \frac{1}{8} + \frac{1}{8} + ...$

$\vdots \quad \vdots \quad \vdots + ...$

$= 1 + \frac{1}{2} \times (1) + \frac{1}{4} \times (1) + ...$

$= 2$

(c) $p(x_0^{n-1}) = p(x_0 x_1 ... x_{n-1})$

$= p(x_0)p(x_0 \to x_1)p(x_1 \to x_2)...p(x_{n-2} \to x_{n-1})$

$p_{x_0} p_{x_0,x_1} p_{x_1,x_2} ... p_{x_{n-2},x_{n-1}}$

(d) Define s_i as the expected number of visits to state i before returning from 0, to state 0. So,

$s_i = \mathbb{E}_0[\sum_{n \geq 1} 1_{\{X_n = i\}} 1_{\{n \leq T_0\}}]$

Where T_0 is when it returns to state 0. And the index 0 of $\mathbb{E}$ shows that we are considering the chain from the time it has left state 0.

Note that

$$\pi(i) = \frac{s_i}{\sum_j s_j}$$

Because,

$$\sum_i \pi(i) p_{ij} = \sum_i \frac{s_i p_{ij}}{\sum_j s_j}$$
$$= \frac{s_j}{\sum_j s_j} = \pi(j)$$

Furthermore, $s_0 = 1$ and $\sum_j S_j = \mathbb{E}(T_0)$ both by definition.

So $\pi(0) = \frac{1}{\mathbb{E}(T_0)}$ which is the answer to the question, not only for the defined extended Markov process, but rather for any general Markov process that has stationary distribution. This is true for any other state as well.

So the answer to the part (d) would be $\frac{1}{p(x_0^{n-1})}$ where $p(x_0^{n-1})$ is calculated in part (c).

(e) $R_n(X_0 X_1 ... X_{n-1}) | (X_0 X_1 ... X_{n-1}) = (x_0 x_1 ... x_{n-1})$ is the distance between the last time the extended Markov state x_0^{n-1} has occurred in the extended Markov process. So by what we defined, the $\mathbb{E}$ of this random variable is exactly the $\mathbb{E}$ of the returning time of the chain from x_0^{n-1} to x_0^{n-1}.

(f) First equality: We encode X_0^{n-1} in binary and this requires $\lceil \log R_n \rceil$ bits. We further send the length of this description ($\lceil \log R_n \rceil$) ans this encoding is done by the code $C(.)$ designed in the hint of the problem. As explained describing k by the code $C(.)$ requires $2\lceil \log k \rceil + 1 < 2logk + 3$ bits and thus to encode $\lceil \log R_n \rceil$, we need a $\log R_n + 2 \log \log R_n + O(1)$ is what the length of communicating R_n which is exactly $l(x_0^{n-1})$.

So

$$\lim_{n\to\infty} \frac{1}{n}\mathbb{E}l(x_0^{n-1}) = \lim_{n\to infty} \frac{1}{n}\mathbb{E}(logR_n + 2loglogR_n + O(1))$$

The second equality is true based on definition of expectation. The third is true because of Jensen's inequality. The forth is again by definition.