The Theory of Computation
Bernard M.E. Moret
Addison-Wesley-Longman, 1998

Solution Manual
Chapter 3, Problem 3.15

1. We may want to write separate expressions for strings of length 0, 1, 2, 3, and for strings of length 4 or more: $\epsilon + 0 + 1 + 01 + 10 + 11 + 010 + 011 + 101 + 110 + 111$ will take care of all permissible strings of length less than 4; with length 4 or more we get $(0+1)^*(0101 + 0110 + 0111 + 1010 + 1011 + 1101 + 1110 + 1111)$; the sum of the two expressions does the trick, even if without any subtlety!
2. We can start by generating all strings that do not have more than two 0s in a row; this can be done in a number of ways, for instance by $(1 + 01 + 001)^*(\epsilon + 0 + 00)$, where the first part always generates something that ends in 1 (unless it is ϵ) and the last part simply allows a last few 0s. Once we have that, we can handle separately strings that may end with four arbitrary characters, remembering that, if the 6th and 5th characters from the end are both 0s, then the tail of four characters cannot start with a 0. The tail is not needed, so we write an ϵ alternative to it. We get $(1 + 01 + 001)^*((\epsilon + 0)(\epsilon + (0 + 1)^4) + 00(\epsilon + 1(0 + 1)^3))$.
3. We can divide the language into strings that do include the triple and strings that do not. The strings with the triple are of the form $x000y$ where x is either ϵ or a string with no triple of 0s that ends with a 1 and y is either ϵ or a string with no triple of 0s that starts with a 1. The key is thus to come up with a regular expression that describes all strings that do not include a triple of 0s, say expression R; then the language is simply $R + (\epsilon + R1)000(\epsilon + 1R)$. We derived an R in the previous problem, although we can be a bit more clever now, since our R was "oriented" (to end with 1) and could have been "oriented" the other way (to start with 1). Using this extra information, we can write $(1 + 01 + 001)^*(\epsilon + 0 + 00) + (1 + 01 + 001)^*000(1 + 10 + 100)^*$, which simplifies to $(1 + 01 + 001)^*(\epsilon + 0 + 00 + 000(1 + 10 + 100)^*)$.
4. We can write a regular expression for this language by brute force: a 0 can be followed by anything and a 1 can be followed by a 0 and

anything. If a 1 is followed by another 1, then 0s cannot appear after that. Thus we can write $(0+10)^*1^*$

5. We assume that the string 000 is considered to contain two pairs of 0s. A simple way to do this problem is to decompose the language into pieces: (i) strings with no pair of consecutive 0s or 1s; (ii) strings with one pair of consecutive 0s and no consecutive 1s; (iii) strings with one pair of consecutive 1s and no consecutive 0s; and (iv) strings with one pair each of consecutive 0s and of consecutive 1s. Part (i) is described by the regular expression

$$(\epsilon + 0)(10)^*(\epsilon + 1)$$

part (ii) is described by the regular expression

$$((\epsilon + 1)(01)^*)00((10)^*(\epsilon + 1))$$

part (iii) is described by the regular expression

$$((\epsilon + 0)(10)^*)11((01)^*(\epsilon + 0))$$

and part (iv) is described by the regular expression

$$((\epsilon+1)(01)^*)00(10)^*11((01)^*(\epsilon+0)) + ((\epsilon+0)(10)^*)11(01)^*00((10)^*(\epsilon+1))$$

The union (sum) of these four expressions is the desired solution.

6. We begin by expressing the language of all strings with no pair of adjacent 0s, call it L_0, and the language of all strings with no pair of adjacent 1s, call it L_1. L_0 can be written as $(1+01)^*(\epsilon+0)$, while L_1 can be written as $(0+10)^*(\epsilon+1)$. Now, our desired language is just the concatenation L_1L_0, or

$$(0 + 10)^*(\epsilon + 1)(1 + 01)^*(\epsilon + 0)$$

Note that we can simplify this expression: the middle $(\epsilon + 1)$ serves no purpose in the concatenation, since the 1 it introduces can also be introduced by the next expression; so we get

$$(0 + 10)^*(1 + 01)^*(\epsilon + 0)$$