The Theory of Computation

Bernard M.E. Moret

Addison-Wesley-Longman, 1998

Solution Manual

Chapter 4, Problem 5.2

1. In order to decrement an unsigned number, we first need to move the head of the machine to the least significant bit of the number, at the right end of the string. We then start moving leftward; if we encounter a 0, we change it into a 1 and loop; if we encounter a 1, we change it into a 0 and (almost) stop; we should not encounter a blank, since this would mean that we were attempting to decrement a value of 0. When we encountered a 1 and changed it into a 0, we should take care that it was not the leading 1, since we would then leave a number with a leading 0; we thus simply check the square to the left: if it is a blank, then we replace our 0 by a blank and stop with the head to the right, otherwise we enter a loop to move the head back to the beginning of the number.
2. We can do this in one pass by keeping track of carries in the finite-state control (or we can use the solution of (3) below and add the number to itself twice). In multiplying by 3, the carry value can vary between 0 and 2. So our machine moves its head to the least significant bit of the input, then starts moving left, in a state that corresponds to a carry of 0. Then, in general, we have the following possibilities:
   - If the carry is 0 and the scanned bit is 0, we write a 0 and keep the carry to 0; if the carry is 0 and the scanned bit is 1, we write a 1 and change the carry to 1.
   - If the carry is 1 and the scanned bit is 0, we write a 1 and change the carry to 0; if the carry is 1 and the scanned bit is 1, we write a 0 and change the carry to 2.
   - If the carry is 2 and the scanned bit is 0, we write a 0 and change the carry to 1; if the carry is 2 and the scanned bit is 1, we write a 1 and keep the carry to 2.

   Finally, if the scanned bit is blank, we simply write any nonzero carry leftward (i.e., we write a 1 if the carry is 1 and we write a 0, then a 1 to the left of it, if the carry is 2).

3. The numbers to be added are separated by some symbol, say a blank; for convenience, we assume that our alphabet includes a fourth symbol, say #. The basic idea is to read the same bit position in each number, add any outstanding carry, and write the new bit of the answer and update the carry, all moving from the least significant bit to the most significant bit—basically the standard longhand addition procedure taught to all schoolchildren. Since we are using binary numbers, the carry value is either 0 or 1; we use the finite-state control to "remember" the values of the carry and of the two scanned bits. To keep track of the current bit position, we simply overwrite the scanned squares with #, the fourth symbol. Since the answer is produced left to right, we write it to the left of the first input string, thereby avoiding the need to keep shifting the output string.

   Our machine will have to do a lot of head motions, so it has two main blocks of states. One block entered after writing the new bit of the answer, simply move the head right until the least non-scanned bit of the second number is encountered—it goes through any 0s and 1s (the answer), one blank, any 0s and 1s (unscanned remainder of first number), zero or more #s, a blank, any 0s and 1s, and stops at a blank or a #, moving right by one. This block exists in two parallel tracks of states, one track for a carry of 0 and one track for a carry of 1. The other block takes it from there and moves left; it reads the current bit (which could be blank if all the second number has been scanned), replaces it by # if it is not a blank, moves left over any other 0s and 1s, one blank, and any #s, scans the next bit (again, could be blank), replaces it by # if not blank, moves left past any 0s and 1s, past one blank, and past any 0s and 1s, and writes the new bit—which is determined by the state, because our machine had several parallel state tracks for this block, corresponding to carry value (0 or 1), bit of second number (0, 1, or blank), and bit of first number (0, 1, or blank). If both bit values are blank, the carry is handled and the machine enters a final block, which will clean up the tape (removing the marked up input strings); otherwise, a new bit is written, the carry updated, and the machine moves right again.