The Theory of Computation

Bernard M.E. Moret

Addison-Wesley-Longman, 1998

Solution Manual
Chapter 4, Problem 5.16

Note that an off-line TM (also equipped with three tapes, except that the work tape is "normal", i.e., can store any alphabet character in any quantity) can do anything that a standard TM can do: just copy the input to the work tape, simulate the standard TM on the work tape, then copy the work tape to the output tape. Thus we can simulate an off-line TM on our model; of course, we can also simulate a RAM or partial recursive functions! In our model, the work tape can be partitioned into four regions by the three pebbles, each region being composed of indistinguishable (empty) cells. The first and last regions (the two ends of the tape) are useless, but the two middle regions are bounded and can be used to count: we can store (in unary, by counting the number of empty cells between two pebbles) two natural integers, call them x and y. By shifting the pebbles left or right, we can increment or decrement either number and we can easily test either one for equality to zero. On the other hand, we can only add destructively (by zeroing one of the operands and replacing the other by the sum) and cannot multiply at all, so that we do not really have enough power to encode and decode. It would take at least 4 pebbles to enable us to do full arithmetic, including multiplication and division by a constant, while retaining values of operands—all of which is needed to encode and decode values. So we have the following alternatives: (i) show that the universal model of computation from which we start can be further restricted into a (still universal) model that is easier to simulate by our pebble machine; or (ii) define a somewhat more powerful version of our pebble machine, show that it can simulate an arbitrary TM/RAM, then show that our 3-pebble machine can simulate the more powerful pebble machine.

First, let us consider a two-register, unary-coded RAM (known to be universal from Exercise 4.15). Simulating such a RAM on our machine is easy: we use our two intervals to store (in unary) the contents of the two registers. The RAM can do one of three things: (i) increment a register and move to the next instruction—we increment the corresponding interval and change state; (ii) decrement its register and move to the next instruction—we decrement the corresponding interval and change state;

and (iii) conditional jump to another instruction—we test an interval for equality to zero by checking whether its two limiting pebbles are adjacent, then change state as needed. The computation is initialized by "copying" the input (which is in unary in register 1) to the work tape (i.e., placing two pebbles to delimit a piece of work tape of the same size as the input); when the simulation is complete, the pebble machine "copies" its first interval to the output tape, that is, it writes one symbol on the output tape for each square between the first two pebbles. Thus we have a complete simulation of a two-register unary-coded RAM on our pebble machine.

Let us consider the alternative of using a slightly more powerful pebble TM. We shall allow ourselves 5 pebbles rather than just 3. With 5 pebbles, we have 4 intervals—4 numbers that can be incremented, decremented, and tested for 0. Now, if we encode the TM's tape in three parts (as we did when simulating a TM on a RAM: left-hand tape, symbol under the head, right-hand tape), we can store the three parts and still have one "scratch" counter. This allows us to manipulate the three parts as needed. Assuming that the original machine can store blanks and *a*s on its tape, we need division by 2, multiplication by 2, and parity test in order to simulate head moves and test for the character under the head. We can divide by two one of our "registers" by initializing the last register to zero (4th and 5th pebble adjacent), then decrementing our register by two for each increment of the scratch register by one; we then copy the result from the scratch register back in to the correct register by the same technique in reverse (increment the register for each decrement of the scratch register); multiplying by two is the same, but we increment twice for each decrement, then copy the result. Parity test is even easier: traverse the register "two squares at a time." These operations allow us to simulate the effect of an arbitrary transition of the original TM on our 5-pebble machine. It remains now to show how a 3-pebble machine can simulate a 5-pebble machine—or how to cut down from 4 registers to just 2 registers. Note that we need only simulate the specific 5-pebble TM produced above on our 3-pebble TM. In particular, the 5-pebble TM produced above always has all 5 pebbles on the work tape—it just shifts them left or right. In our simulation on the 3-pebble machine, we will keep the contents of the four registers of the 5-pebble machine encoded as a single number (in unary, naturally) in our first interval; the second will be used for temporaries. We proceed exactly as for the reduction of an arbitrary RAM to a two-register RAM described in Exercise 4.15: if the four registers of our 5-pebble machine contain values i, j, k, and l, we will encode them in our first interval as the single number $2^i \cdot 3^j \cdot 5^k \cdot 7^l$. Incrementation, decrementation, and test for zero are all as described earlier for the RAM simulation.