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

Solution Manual
Chapter 4, Problem 5.21

What Exercise 4.20, part (4), demonstrated is that, when we can restrict the form of the numbers processed by a system of rules, we can control fairly easily the sequencing of the rules. Sequencing is of course crucial in simulating a RAM program or Turing machine with a Post system; the other crucial aspect is testing for the absence of something—but again, we showed how to do that in Exercise 4.20, part (4), with the block of rules that added a factor of 13 only to a number that did not have such a factor.

Say we want to simulate a RAM program. We shall encode everything with prime power encoding. That is, the initial state of the RAM (contents of registers) will be a number of the form $2^{i_1}3^{i_2}5^{i_3}\cdots$, where i_1 is the value of the first register, i_2 the value of the second register, etc. In addition, this value will be multiplied by a prime corresponding to the value of the program counter (we can number each line in the RAM program starting at 1)—since there are only a constant number of lines in any given program, this uses only a constant number of primes. Each instruction in the program will be encoded by a rule or a block of rules, guarded by the value of the program counter, so that it can only be applied when the program has reached that line. Thus rules corresponding to a RAM program instruction numbered, say 17, will in general be of the form $p_{17}\cdot(\ \ )\longrightarrow p_{17}\cdot(\ \ )$, where p_{17} is the prime assigned to the 17th line of code in the RAM program. (Overall, then, we need at least one prime per line and one prime per register; we will also need some additional primes to play tricks of the type used in Exercise 4.20, part (4).) Our most elementary RAM has only three instructions: increment a register, decrement a register, and jump on zero to an address. Increment and decrement simply remove or add one factor and increase the program counter; for instance, the instruction at line 20

```
20:  inc R3
```

becomes the rule

$$p_{20}\cdot x\longrightarrow p_{21}\cdot 5\cdot x$$

since 5 is the prime assigned to register 3. Decrement is just a bit trickier, because it appears we need a test for 0, since decrementing a 0 value leaves 0. However, one slick way of handling this problem is to encode a value of i in register, say 3, by the value 5^{i+1}, thereby giving ourselves an extra factor and making the test for the absence of a factor much simpler. The jump on zero is similar: we need a test for the absence of a factor; in both cases, the prime denoting the program counter is replaced by another, but which other depends on that test.

The details are somewhat long and not much different from what we did in Exercise 4.20, part (4), so we omit them here.