Chapter 12

Computable Functions

12.1. Suppose this function f is computable, and let T_f be a TM with tape alphabet $\{0,1\}$ computing f. Let $T = T_f T_1$, where T_1 is a TM that moves its tape head to the first blank square to the right of its starting position and halts. Suppose T has n states. Then on input n, T makes at least $f(n)+1$ moves moves before halting, since T_1 moves its tape head $f(n)+1$ times. However, by definition of f, no TM with n states and tape alphabet $\{0,1\}$ can make more than $f(n)$ moves, when started with input 1^n. This is a contradiction.

12.2. Suppose f is computable. Then $g = f+1$ is also. Let T_g be a TM computing g, and suppose that T_g has n states and m tape symbols. By adding useless states if necessary, we can assume $n \geq m$. Then on input 1^n, T_g leaves output $1^{f(n)+1}$, which by definition contains more 1's than any TM with n states and m tape symbols can possibly leave on the tape, starting with input 1^n. This is a contradiction.

12.3. For any integer n, there are only a finite number of TMs having n nonhalting states and tape alphabet $\{0,1\}$. If the halting problem were solvable, then we would be able to eliminate all those that did not halt on the input 1^n. We could then compute $bb(n)$ by running each of the remaining ones on this input and seeing which one printed the most 1's.

12.4. For any n, we can construct the following TM T_n. When started with a blank tape, T_n halts in the accepting state with tape $\underline{\Delta}1^n$. Furthermore, it is easy to see that the number of states required for T_n is essentially n: more precisely, $n+k$, for some fixed k independent of n. T_n requires no tape symbols other than 1. Now let f be any computable function, and let T_f be a TM with m states and tape alphabet $\{0,1\}$ that computes f. Let T'_n be the composite TM that first executes T_n, starting with a blank tape, and then executes T_f. The number of states of T'_n is $k'+n$ (where k' is a constant independent of n), and the output produced by T'_n, assuming it begins with a blank tape, is $1^{f(n)}$. It follows from the definition of bb that $bb(k'+n) \geq f(n)$.

Suppose for the sake of contradiction that bb is computable. Then so is the function $bb(2n)$. According to the preceding paragraph, $bb(k'+n) \geq bb(2n)$ for every n. But this is clearly not correct, because for sufficiently large n, there are TMs with $2n$ states and tape alphabet $\{0,1\}$ which, when started with a blank tape, can write more 1's before halting than any such TM having $n+k'$ states.

12.5. It is clear that if f is computable, then the problem: Given n and C, is $f(n) > C$? is solvable. On the other hand, if this problem is solvable, then for any n, $f(n)$ is the smallest value of C for which $f(n) \not> C$, and so there is an obvious algorithm for computing f.

12.6. *PR* would be unchanged. All constant functions of 0 variables can be obtained from the successor function and C_0^0 by repeated applications of composition. Moreover, the

constant function C_k^{n+1} can be obtained from C_k^n and p_{n+2}^{n+2} by primitive recursion. It follows by mathematical induction that all constant functions are primitive recursive according to this definition.

12.7. The only new functions would be the functions f of one variable having the formula $f(x) = x + m$ (for some $m \geq 1$) and the functions g of several variables having the formula $g(x_1, x_2, \ldots, x_k) = x_i + m$ (for some i with $1 \leq i \leq k$ and some $m \geq 1$.

12.9. $f_2(x)$ is 1 if $x = 0$ and 0 otherwise. $f_6(x) = Mod(x, 2)$. $f_{14}(x, y) = Mod(x, 2) + y$. $f_{15}(x) = Div(x, 2)$.

12.10. $g = 0 = C_0^0$; $h(x, y) = y + 2x + 1$.

12.11. (a)
$$\begin{aligned} f_0 &= p_1^1 \\ f_1 &= s \text{ (the successor function)} \\ f_2 &= p_3^3 \\ f_3 &= s(p_3^3) \text{ (obtained from } f_1 \text{ and } f_2 \text{ by composition)} \\ f_4 &= Add \text{ (obtained from } f_0 \text{ and } f_3 \text{ by primitive recursion)} \\ f_5 &= Add(Add, Add) \text{ (obtained from } f_4 \text{ by composition)} \\ f_6 &= p_1^2 \\ f_7 &= f \text{ (obtained from } f_4, f_5, \text{ and } f_6 \text{ by composition)} \end{aligned}$$

(c)
$$\begin{aligned} f_0 &= 1 = C_1^0 \\ f_1 &= s \text{ (the successor function)} \\ f_2 &= s(s) \text{ (obtained from } f_1 \text{ and } f_1 \text{ by composition)} \\ f_3 &= C_0^0 \\ f_4 &= p_2^2 \\ f_5 &= s(s(p_2^2)) \text{ (obtained from } f_2 \text{ and } f_4 \text{ by composition)} \\ f_6 &= 2x \text{ (obtained from } f_3 \text{ and } f_5 \text{ by primitive recursion)} \\ f_7 &= 2p_2^2 \text{ (obtained from } f_6 \text{ and } f_4 \text{ by composition)} \\ f_8 &= f \text{ (obtained from } f_0 \text{ and } f_7 \text{ by primitive recursion)} \end{aligned}$$

(e) Use the fact that $|x - y| = (x \dot{-} y) + (y \dot{-} x)$.

12.12. We prove the result for Add_n by mathematical induction on n, and the result involving *Mult* is similar. The basis step follows by observing that $Add_1(x_1) = p_1^1(x_1)$. Assume that $k \geq 0$ and that Add_k is a primitive recursive function of k variables. We wish to show that Add_{k+1} is a primitive recursive function of $k + 1$ variables. This follows from the formula

$$Add_{k+1}(x_1, \ldots, x_{k+1}) = Add(Add_k(x_1, \ldots, x_k), x_{k+1})$$

and Theorem 13.1.

12.14 For each i satisfying $0 \leq i \leq n_0 + p - 1$, let $m_i = f(i)$. Then $f(n)$ can be defined by

cases as follows. For each $i < n_0$, $f(n) = m_i$ if $n = i$. For each i with $n_0 \leq i \leq n_0 + p - 1$, $f(n) = m_i$ if $n \geq n_0$ and $Mod(n, p) = Mod(i, p)$. Since the cases are all defined by primitive recursive predicates, it follows that f is primitive recursive.

12.15. (a) f can be obtained by primitive recursion from the two functions p_1^1 and h, where h is defined by the formula

$$h(x, y, z) = \begin{cases} x & \text{if } x > y \\ y + 1 & \text{if } x \leq y \end{cases}$$

(c) $f(n) = \min\{x \leq n + 1 \mid x^2 > n\} \dot{-} 1$.

12.17. From Lemma 12.1, we know that the function f_1 defined by

$$f_1(x, k) = \sum_{i=0}^{k} g(x, i)$$

is primitive recursive. Since $f(x) = f_1(x, x)$, f is also.

12.18. This follows from the formula

$$HighestPrime(k) = \max\{y \leq k \mid Exponent(y, k) > 0\}$$

together with the next exercise.

12.19. (a) We have $m^P(X, 0) = 0$ and

$$m^P(X, k + 1) = \begin{cases} k + 1 & \text{if } P(X, k + 1) \\ m^P(X, k) & \text{if } \neg P(X, k + 1) \end{cases}$$

It follows that m^P is primitive recursive.

(b) $m^P(X, k)$ can be expressed as

$$\begin{aligned} & \min\{y \leq k \mid \text{ for every } z \text{ with } y < z \leq k, \neg P(X, z)\} \\ = \ & \min\{y \leq k \mid \text{ for every } z \leq k, z \leq y \vee \neg P(X, z)\} \end{aligned}$$

Thus m^P can be expressed in terms of a bounded minimalization.

12.20. If $H(x, y)$ is the predicate described in the chapter,

$$H(x, y) = \text{ there exists } y \text{ so that } T_u \text{ halts after exactly } y \text{ moves on input } s_x$$

then we may let $N(x, y)$ be the negation of $H(x, y)$. The unbounded universal quantification of N is the 1-place predicate

$$\text{for every } y, T_u \text{ fails to halt after } y \text{ moves on input } s_x$$

and this is not computable for the same reason that $Halts(x)$ is not.

12.21. We can take $f(X, y)$ to be $1 - \chi_P(X, y)$.

12.23. The bounded operations are special cases of the unbounded. For example, suppose P is an $(n+1)$-place predicate, and define the $(n+2)$-place predicate P_1 by

$$P_1(X, y, z) = (P(X, z) \text{ or } z = y + 1)$$

Then the unbounded minimalization of P_1 is the partial function M_{P_1} of $n+1$ variables defined by

$$M_{P_1}(X, y) = \min\{z \mid P_1(X, y, z)\}$$

If there is a z satisfying $0 \leq z \leq y$ and $P(X, z)$, then $M_{P_1}(X, y)$ is the smallest such z, and otherwise $M_{P_1}(X, y)$ is $y + 1$. In either case, $M_{P_1}(X, y) = m_P(X, y)$.

12.24. If there were a solution to this problem, then in particular there would be a solution to the problem Given a TM computing some partial function, does it halt on every input? Now we know that the (apparently) more general problem Given an *arbitrary* TM, does it accept every input? is unsolvable. But it follows from this that the more restricted problem is also unsolvable, since if T is any TM, there is another TM T' that accepts exactly the same strings as T and computes some partial function. (T' can be constructed as follows: first modify T so that instead of writing Δ it writes a different symbol, say Δ'; then let $T' = TT_1$, where T_1 erases the tape and halts with the tape head on square 0. T' halts on input x if and only if T does, and T' computes the constant partial function whose value is the null string.) Therefore, the given problem is unsolvable.

12.25. We can use Theorem 12.9 with $n = 0$, provided we can come up with an appropriate function $h : \mathcal{N}^2 \to \mathcal{N}$. We are looking for a formula of the form

$$f(k+1) = h(k, gn(f(0), f(1), \ldots, f(k)))$$

where h is primitive recursive. For simplicity we ignore the fact that $h(p, q)$ will have to be defined for $p = 0$ separately (since $f(1)$ is not defined using the recursive formula).

The recursive formula for f can be written $f(k+1) = 1 + f(\lfloor\sqrt{k+1}\rfloor)$. The point is that $\lfloor\sqrt{k+1}\rfloor$ is one of the numbers $f(0), f(1), \ldots, f(k)$. So intuitively, the function h is supposed to produce one of the $k+1$ numbers, the Gödel number of which is the second parameter. Remembering that $f(i) = Exponent(i, gn(f(0), \ldots, f(k)))$, let us write

$$h(p, q) = 1 + \sum_{i=0}^{p} c_i(p) Exponent(i, q) = 1 + \sum_{i=0}^{p} d_i(p, q)$$

where we want $c_i(p)$ to be 0 for all values of i except one (the i for which $\lfloor\sqrt{p+1}\rfloor = i$) and to be 1 for that i.

It is not hard to see that each c_i can be defined by considering two cases, each of which is described by a primitive recursive predicate. It follows from Exercise 12.17 (actually, the

generalization of this exercise to functions of two variables) that $h(p, q)$ will be primitive recursive, since each of the functions d_i is.

12.26. Let $g(x, y) = |f(y) - x|$. Then $f^{-1}(x) = M_g(x) = \mu y[g(x, y) = 0]$.

12.27. As in the chapter, we denote the symbols of the alphabet by $a_1, \ldots a_s$. Then an integer x is the Gödel number of a string of length m if and only if

$$x = 2^{i_0}3^{i_1} \ldots PrNo(m-1)^{i_{m-1}}$$

where each of the exponents i_j satisfies $1 \leq i_j \leq s$. Therefore, $Isgn(x)$ if and only if there is an integer m so that for every i satisfying $0 \leq i \leq m \dot{-} 1$, $1 \leq Exponent(i, x) \leq s$, and for every $i \geq m$, $Exponent(i, x) = 0$. This predicate is the existential quantification of another predicate, and the other predicate involves two universal quantifications. Since $Exponent(i, x) = 0$ for every $i \geq x$, we may express all these quantifications as bounded ones. $Isgn(x)$ is therefore equivalent to this statement: there exists $m \leq x$ so that for every $i \leq m \dot{-} 1$, $1 \leq Exponent(i, x) \leq s$ and for every $i \leq x$, if $i \geq m$ then $Exponent(i, x) = 0$. It follows that $Isgn$ is a primitive recursive predicate.

12.28. (a) $f : \Sigma^* \to \mathcal{N}$ is primitive recursive if the corresponding function $\tau_f : \mathcal{N} \to \mathcal{N}$ is, where τ_f is defined by

$$\tau_f(n) = f(gn'(n))$$

(gn' being the "left inverse" of gn discussed in Section 13.6). The definition for μ-recursive is similar.

(b) If $f(x) = |x|$, then

$$\tau_f(n) = |gn'(n)| = \begin{cases} 0 & \text{if NOT } Isgn(n) \\ HighestPrime(n) + 1 & \text{if } Isgn(n) \end{cases}$$

It is clear from this formula that τ_f, and thus f, is primitive recursive.

12.29. Let g be a computable total function. Then $h = g + 1$ is also. Let T_h be a TM with tape alphabet $\{0, 1\}$ computing h, with k states. For any n, $h(n)$ is the number of 1's left on the tape by T_h if it begins with input 1^n; therefore, $h(n)$ is no larger than the maximum number of 1's left on the tape by any TM having k states and tape alphabet $\{0, 1\}$, if it starts with input 1^n and eventually halts. Clearly, if $n \geq k$, this number is no larger than the maximum number of 1's left on the tape by any TM having n states and tape alphabet $\{0, 1\}$, if it starts with input 1^n and eventually halts. This last number is the definition of $f(n)$, from which it follows that $g(n) < h(n) \leq f(n)$ for every $n \geq k$.

12.31. (a) f can be obtained by primitive recursion from the two functions g and h, defined as follows.

$$g(x) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{if } x > 0 \end{cases}$$

$$h(x,y,z) = \begin{cases} s(z) & \text{if } Mod(x, k+1) = 0 \\ z & \text{if } Mod(x, k+1) \neq 0 \end{cases}$$

Prime can now be defined in terms of f:

$$Prime(n) = (n \geq 2 \wedge f(n, n\dot{-}1) = 1)$$

12.32. (b) We begin with the observation that the last digit in the decimal representation of an integer n is $Mod(n, 10)$. It is therefore sufficient to show that the function g is primitive recursive, where $g(0) = 1$, $g(1) = 14$, $g(2) = 141$, and so on. $g(i)$ is the greatest integer less than or equal to the real number $10^i\sqrt{2}$. Equivalently, $g(i)$ is the largest integer whose square is less than or equal to $(10^i * \sqrt{2})^2 = 2 * 10^{2i}$. This means that $g(i)+1$ is the smallest integer whose square is greater than $2 * 10^{2i}$. Therefore, if we apply the minimalization operator to the primitive recursive predicate P defined by $P(x,y) = (y^2 > 2 * 10^{2*x})$, $g(x)$ is obtained from subtracting 1 from the result. The only remaining problem is to show how a *bounded* minimalization can be used in this last step. We can do this by specifying a value of y for which y^2 is certain to be greater than $2 * 10^{2*x}$. Clearly, $y = 2 * 10^x$ is such a value, since $(2 * 10^x)^2 = 4 * 10^{2x} > 2 * 10^{2x}$. Therefore, $f(x) = m_P(x, 2 * 10^x)\dot{-}1$.

12.33. We give the proof for the product; the proof for the sum is simpler. First we recall that the empty product (corresponding to the case when $l(k) > m(k)$) is taken to be 1. Assuming that $l(k) \leq m(k)$, we could write

$$f_1(X,k) = Div(\prod_{i=0}^{m(k)} g(X,i), \prod_{i=0}^{l(k)\dot{-}1} g(X,i))$$

—except for two potential problems: the first is the possibility of dividing by 0 (although our formula is defined in that case, it doesn't produce the correct value), and the second is that $l(k)$ may be 0. To get around the first problem, we define

$$g'(X,i) = \begin{cases} g(X,i) & \text{if } g(X,i) \neq 0 \\ 1 & \text{if } g(X,i) = 0 \end{cases}$$

It can then be verified easily that

$$f_1(X,k) = \begin{cases} 1 & \text{if } l(k) > m(k) \\ 0 & \text{in case II} \\ \prod_{i=0}^{m(k)} g(X,i) & \text{in case III} \\ Div(\prod_{i=0}^{m(k)} g'(X,i), \prod_{i=0}^{l(k)\dot{-}1} g'(X,i)) & \text{otherwise} \end{cases}$$

where case II is when there exists i with $l(k) \leq i \leq m(k)$ and $g(X,i) = 0$, and case III is when this condition fails and $l(k) = 0$. Moreover, this formula makes it clear that f_1 is

primitive recursive. (Note that the predicate in case II is primitive recursive, since it can be expressed in terms of bounded minimalization.)

12.34. A valid μ-recursive derivation involves the application of unbounded minimalization only to total functions. Since it may not be possible to determine whether a given function is total (see Exercise 12.24), it may not be possible to determine whether a string purporting to be a μ-recursive derivation is in fact valid.

12.35. (b) If $f(x) = ax$, then the function ρ_f takes the number

$$2^{i_0}3^{i_1}\ldots PrNo(m)^{i_m}$$

to the number

$$2^1 3^{i_0} 5^{i_1}\ldots PrNo(m+1)^{i_m}$$

In other words, ρ_f is defined by the formula

$$\rho_f(n) = 2 * \prod_{i=0}^{r(n)} g(n,i)$$

where $r(n) = HighestPrime(n)$ (see the proof of Lemma 12.4) and $g(n,i) = PrNo(i+1)^{Exponent(i,n)}$. This is almost, but not quite, in the form of Lemma 12.1. According to that result, the function

$$h(n,y) = \prod_{i=0}^{y} g(n,i)$$

is primitive recursive. But $\rho_f(n) = 2 * h(n, r(n))$, from which it follows that ρ_f is also.

(c) The argument is similar to that in part (b). In this case we may write

$$\rho_f(n) = \prod_{i=0}^{r(n)} g(n,i)$$

where $r(n) = HighestPrime(n)$ and

$$g(n,i) = (PrNo(i))^{Exponent(HighestPrime(n) \dot{-} i, n)}$$

12.36. (a) $f : (\Sigma^*)^n \to \Sigma^*$ is primitive recursive (resp. recursive) if the corresponding function $\pi_f : \mathcal{N}^n \to \mathcal{N}$ is, where π_f is defined by

$$\pi_f(n_1,\ldots,n_k) = gn(f(gn'(n_1),\ldots,gn'(n_k)))$$

(b) If f is the concatenation function, then $\pi_f : \mathcal{N} \times \mathcal{N} \to \mathcal{N}$ is defined by

$$\pi_f(m,n) = gn(gn'(m)gn'(n))$$

$$= m * \prod_{i=l(m,n)}^{u(m,n)} h(m,n,i)$$

where

$$\begin{aligned} l(m,n) &= \mathit{HighestPrime}(m) + 1 \\ u(m,n) &= \mathit{HighestPrime}(m) + \mathit{HighestPrime}(n) + 1 \\ h(m,n,i) &= \mathit{PrNo}(i)^{\mathit{Exponent}(i \dot{-} \mathit{HighestPrime}(m) \dot{-} 1, n)} \end{aligned}$$

It follows (see the solution to Exercise 12.35) that π_f is primitive recursive.