

Homework 6 Solutions, CS 132, Winter 2005

March 10, 2005

12.10 Find two functions g and h so that the function f defined by $f(x) = x^2$ is obtained from g and h by primitive recursion.

$f(x) = x^2$ can be written recursively as $f(0) = 0$ and $f(x + 1) = f(x) + 2x + 1$. So we can use the following functions to create a primitive recursive definition:

$$f(0) = g = 0$$

$$f(k + 1) = h(k, f(x))$$

Where h and g are defined as

$$g = C_0^0$$

$$h(x, y) = y + 2x + 1 = \text{Add}(\text{Add}(y, \text{Mult}(2, x)), 1) = \text{Add}(\underbrace{\text{Add}(P_2^2(x, y), \text{Mult}(\underbrace{C_2^2(x, y)}_2, \underbrace{P_1^2(x, y)}_x))}_y, \underbrace{C_1^2(x, y)}_1)$$

Add and Mult are shown to be primitive recursive in chapter 12.

Here is an example with actual numbers:

$$f(2 + 1) = h(2, f(2)) = h(2, h(1, f(1))) = h(2, h(1, h(0, f(0))))$$

Now we can plug in our definitions of h and g

$$\begin{aligned} h(2, h(1, h(0, f(0)))) &= h(2, h(1, h(0, g))) = h(2, h(1, 0 + 2 \cdot 0 + 1)) \\ &= h(2, h(1, 1)) = h(2, 1 + 2 \cdot 1 + 1) = h(2, 4) = 4 + 2 \cdot 2 + 1 = 9 \end{aligned}$$

Which is good, since $3^2 = 9$.

12.26 Suppose that $f : \mathcal{N} \rightarrow \mathcal{N}$ is a μ -recursive total function that is a bijection from \mathcal{N} to \mathcal{N} . Show that its inverse f^{-1} is also μ -recursive.

This function is $f^{-1}(y) = x$ such that $f(x) = y$; note that since f is a bijection, f^{-1} is well defined. Let

$$g(x, y) = (f(y) \dot{-} x) + (x \dot{-} f(y)) = |f(y) - x|.$$

g is primitive recursive since it involves only the primitive recursive functions $+$ and $\dot{-}$. We can then define f^{-1} as

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

13.9 Find the time complexity for each of these TMs:

a. The TM in Example 9.2 accepting the language of palindromes over $\{0, 1\}$.

$$\frac{n^2 + 3n + 4}{2} = O(n^2)$$

b. The copy TM shown in Figure 9.12.

$$2n^2 + 4n + 3 = O(n^2)$$

- 13.10 Show that if L can be recognized by a TM T with a doubly infinite tape, and $\tau_T = f$, then L can be recognized by an ordinary TM with time complexity $O(f)$.

First we need to show that for any machine $T = (Q, \Sigma, \Gamma, q_0, \delta)$ with a doubly-infinite tape, there is a TM $T' = (Q, \Sigma', \Gamma', q_0, \delta')$ with a singly-infinite tape such that $L(T) = L(T')$. Recall that the doubly-infinite machine has a tape where cell-indices to the left of the starting position are negative and to the right of the starting position are positive. Our construction is similar to the one used in Chapter 9 for the two tape TM: we use the alphabet to encode the symbol at cell i and $-i$ on the tape of T in a single symbol at cell i on the tape of T' .

To do this we expand the alphabet so that Γ' includes a symbol for every tuple in $\Gamma \times \Gamma \times \{\Delta\}$; where the tuple γ_1, γ_2 at cell i on the tape of T' indicates that the tape of T would contain γ_1 at $-i$ and γ_2 at i .

Q' contains a state for every tuple in $Q \times \{+, -\}$, where the plus and minus indicate what side of the tape T' is currently operating on.

δ' is constructed so that if $\delta(q, a) = (r, b, R)$ then $\delta'(q+, ac) = (r+, bc, R)$ and $\delta'(q-, ac) = (r-, ad, L)$. Cell 0 is marked by $\#$, a symbol not in the alphabet. This symbol causes T' to switch from the positive to the negative side of the tape using transitions: $\delta(q+, \#) = (q-, \#, R)$ and $\delta(q-, \#) = (q+, \#, R)$.

Once we have this construction determining the time complexity is easy. Note that T' makes only a single transition for each transition in T plus additional transitions for adding the marker $\#$ and reversing directions. These extras are only constant factors, however, so if T has time complexity f then T' has complexity $O(f)$.

- 12.29 Let $b : \mathcal{N} \rightarrow \mathcal{N}$ be the busy beaver function. Show that b is eventually larger than any computable function; in other words, for any computable total function $g : \mathcal{N} \rightarrow \mathcal{N}$ there is an integer k so that $b(n) > g(n)$ for every $n \geq k$.

We can show this by contradiction. To derive the contradiction, assume that there is some function which is not eventually less than the busy beaver function; that is, assume there is some g such that for any integer k there exists an $n \geq k$ such that $b(n) \leq g(n)$.

Since g is, by assumption, computable, the function $h(n) = g(n) + 1$ is also computable, and there is some TM T_h that computes it. Let m be the number of states in T_h . Note that for any $n > m$ we can pad T_h with unreachable states to create an n state TM $T_{h,n}$ which also computes h .

By our assumption there is some $n > m$ such that $h(n) > b(n)$, and thus on input 1^n h writes more than $b(1^n)$ 1s. This implies that there is also an n state machine $T_{h,n}$ that writes more than $b(n)$ 1s. However by the definition of b no n -state TM writes more than $b(n)$ 1s. This is a contradiction and we can conclude that b is eventually larger than all computable functions.