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

Solution Manual
Chapter 5, Problem 5.10

There are several ways to attack this problem. Perhaps the simplest way is to use the step function that was defined for Turing machines.

The $\{\phi_i\}$ programming system that we developed to simulate a Turning machine already has a step function that is very close to what we need. We need only modify it to make it start from the initial state and to check whether the Turing machine has reached the halt state after m steps; if it has, we clean up the tape as described in the text and return the output value (plus 1), otherwise we return ε. We need to translate back and forth, which requires universal and s-1-1 functions on each side—which we have, since we are working with acceptable programming systems. So, to get step(i, x, m) in our $\{\psi_j\}$ system, we translate i into the $\{\phi_i\}$ system, obtaining say $t(i)$, then define our modified Turing machine step function in terms of $t(i)$, x, and m (where m is taken to be a number of steps) to obtain $\text{step}_{TM}(t(i), x, m)$, which is some (total) function ϕ_k in the TM system; we then translate back from ϕ_k to the $\{\psi_j\}$ system, obtaining some (total) function $\psi_{t'(k)}$. Now, by construction, we have $\psi_{t'(k)}(< t(i), x, m >) = \text{step}_{TM}(t(i), x, m)$, which returns either ε or $\text{Succ}(\phi_{t(i)}(x)) = \text{Succ}(\psi_i(x))$. It is easily verified that, whenever $\psi_i(x)$ is defined, there must exist some value m_0 of m (the number of steps required by TM $t(i)$ to stop on input x) such that $\psi_{t'(k)}(< t(i), x, m_0 >) = \text{Succ}(\psi_i(x))$, whereas otherwise $\psi_{t'(k)}(< t(i), x, m >) = \varepsilon$ for all m. Now we write formally $\text{step}(< i, x, m >) = \psi_{c(t'(k),l}(< i, x, m >)$, where c is the composition function and l is the index of the total recursive function $\psi_l(z) =< t(\Pi_1(z)), \Pi_2(z) >$.

In a second approach, we can just take the Turing machine program that implements the step function and encode it directly into our $\{\psi_j\}$ system rather than consider its encoding into the $\{\phi_i\}$ system and translate back and forth.

A more ambitious approach remains purely internal to our programming system. Suppose that, for each fixed m, we defined a new, bounded version of μ-recursion as follows: given function f of $n+1$ arguments, we define the function g of n arguments by m-bounded μ-recursion as the successor of the smallest z, $z \leq m$, such that $f(z, x_1, \ldots, x_n) = \epsilon$; if no z

is found that meets the criteria, g returns ϵ. Because the search on z is bounded by m, it always terminates. Thus, provided that f is total, so is g. Any number of applications of m-bounded μ-recursion construct only total functions. If the original (unbounded) μ-recursion terminates on arguments $x_1, \ldots, x_n$, then there must exist an m (the value returned in the original μ-recursion, in fact) such that the m-bounded recursion finds a z_0 meeting the predicate and thus returns $\mathrm{Succ}(z_0)$, whereas, if the original μ-recursion would have produced an undefined result, then the m-bounded μ-recursion always returns ε. If we replace μ-recursion by m-bounded μ-recursion in the definition of ψ_j in our system, then correct for the use of *Succ* and the ε flag, to propagate ε throughout and to remove the Succ as needed, we shall obtain a new recursive function, call it f_j^m. Whenever $f_j^m(x) \neq \varepsilon$, then $\psi_j(x)$ is defined and $f_j^m(x) = \mathrm{Succ}(\psi_j(x))$. Note that everything remains properly defined within our original programming system, simply because bounded μ-recursion is not really a new definitional scheme, but a higher-level composition operator that can be defined in terms of the original primitive recursive base functions and schemes.

So now we need only show that there is a single recursive function $\mathrm{step}(< i, x, m >)$ such that $\mathrm{step}(< i, x, m >) = f_j^m(x)$ and we will be done. But recall that the index i is the coding of the sequence of applications of the three construction schemes to the base functions that define the function ϕ_i; we can decode this index i to obtain the various pieces and replace each application of μ-recursion by an application of m-bounded μ-recursion (which has its own code, since we can define it through primitive recursion), then we can recode the complete index to obtain the index of f_j^m within $\{\psi_j\}$; finally, we can invoke ψ_{univ} to execute the function given by this index.