1 The Kleene $T$ Predicate

We have already defined $\text{Init}_M(x)$ and $\text{Next}_M(w)$, where

\[
w = \langle \text{state}, \langle \text{symbols to the right} \rangle, \langle \text{symbols to the left} \rangle \rangle .
\]

And furthermore, we have defined the predicate $\text{Comp}_M(x,v)$. Recall that

\[
\text{Comp}_M(x,v) \iff v \text{ is a sequence } \langle v_0, \ldots, v_{l-1} \rangle,
\]

where

\[
v_0 = \text{Init}_M(x),
\]

\[
v_{i+1} = \text{Next}_M(v_i),
\]

\[
v_{l-1} = \text{halting configuration}
\]

We now define the Kleene $T$ predicate. This predicate says something like $\text{Comp}_M(x,v)$, but without fixing the Turing machine $M$. $T(e,x,w)$ means “$w$ codes a complete computation of the Turing machine $M$ with G"odel number $\uparrow M \uparrow = e$ on input $x$.” We claim that this is primitive recursive. (Note that the reason why this might be dubious is that $\uparrow M \uparrow$ might not be primitive recursive.)

One way to prove this would be to create a new Next function which takes in $\uparrow M \uparrow$ and $x$ and gives the next configuration.

We show that $T$ is primitive recursive another way. Define

\[
f(e,x) = \text{output}(\mu w T(e,x,w)) ,
\]

where

\[
\text{output}(w) = \begin{cases} 
\text{value output by TM in configuration } w \text{ if it's in state } q_H \\
0 \text{ otherwise}
\end{cases}
\]

and $\mu w \ldots$ means “the least $w$ such that $\ldots$”. Notice that the output function is primitive recursive.

**Theorem 1.** For any partial recursive function $g(x)$ there is an $e \in \mathbb{N}$ such that $\forall x \in \mathbb{N}, \ g(x) = f(e,x) \text{ and } g(x) = \text{output}(\mu w T(e,x,w))$.  

Proof. Let \( g \) be computed by some Turing machine \( M \). Let \( e = \langle M \rangle \). Now the result follows from applying the appropriate definitions.

Now since the output function is primitive recursive, \( \mu \) is primitive recursive, and \( g \) is primitive recursive, we have the desired result: \( T \) is primitive recursive as well.

## 2 Some Remarks on Unbounded Minimization

Let \( h_2(x\bar{y}) = (\mu z)(R(z, \bar{y})) := \begin{cases} \text{least } y \text{ s.t. } R(z, y) \text{ if it exists} \\ \text{undefined otherwise} \end{cases} \). We define an algorithm for (partially) computing \( h_2(\bar{y}) \):

1. Input \( \bar{y} \).
2. Loop: \( z = 0, 1, 2, \ldots \)
   - Evaluate \( R(z, \bar{y}) \).
   - If accepts, then output \( z \)
   - End loop.

This algorithm proves the following theorem.

**Theorem 2.** If \( R(z, y) \) is recursive, then \( h_2(\bar{y}) \) is partial recursive.

Now we present another kind of unbounded minimization. Let \( h_3 \) be a partial recursive function. Then define \( h_4(y) = (\mu z)(h_3(z, \bar{y}) = 0) \). Here’s an algorithm for \( h_4 \):

1. Take input \( y \).
2. Loop \( z = 0, 1, 2, 3, \ldots \)
   - Evaluate \( h_3(z, \bar{y}) \).
   - If this halts and outputs 0, then output \( z \)
   - End loop.

So we have:

\[
 h_4(y) = (\mu z)(h_3(z, \bar{y}) = 0) := \begin{cases} z \text{ s.t. } h_3(z, \bar{y}) = 0 \text{ and } \forall z' < z, h_3(z', \bar{y}) \downarrow \neq 0 \text{ if there is such a } z \\ \text{undefined otherwise} \end{cases}
\]

And we have the following theorem and corollary.

**Theorem 3.** \( h_4 \) is partial recursive.
Corollary 1. For \( e \in \mathbb{N} \), \( g(x) = \text{output}(\mu w \ T(e, x, w)) \) is partial recursive.

Note that unbounded minimization takes us out of the realm of primitive recursive.

3 Runtime and Primitive Recursive Runtime

We begin with some definitions.

Definition 1. A Turing machine \( M \) has runtime \( s(n) \) for \( s : \mathbb{N} \to \mathbb{N} \) if for all \( x \in \mathbb{N} \) (or \( x \in \Sigma^* \)), if \( n = |x| \) (where \( |x| \) is the length of \( x \), or number of symbols in \( x \)) then \( M(x) \) runs for \( \leq s(n) \) steps.

Definition 2. Furthermore, if \( s(n) \) is primitive recursive then \( M \) is said to have primitive recursive runtime.

To conclude, we prove one little theorem about Turing machines with primitive recursive runtime.

Theorem 4. If \( f \) is a function computed by a Turing machine with primitive recursive runtime, then \( f \) is primitive recursive.

Proof. Let \( M \) compute \( f \). Then we know

\[
  f(x) = \text{output}(\mu w \leq \text{Bd}(s(|x|)) \text{ s.t. } T(\uparrow M \downarrow, x, w)),
\]

where \( \text{Bd}(s(|x|)) \) upper bounds the \( w \)'s that code \( s(|x|) \) steps of a Turing machine.

Now note that the \( \text{Bd} \) function is primitive recursive. So everything on the right hand side is primitive recursive, and hence \( f \) is as well. \( \square \)