Problem. Sketch an example of a Turing machine for implementing primitive recursion (i.e., a for-loop), the way we did it in class, on the example of the following simple for-loop

\[
v = a;
for(int i = 1; i <= b; i++)
\{v = v + i;\}
\]

No details are required, but any details will give you extra credit.

Solution. In mathematical terms, the above for-loop takes the following form:

\[
v(a, 0) = a;
\]

\[
v(a, m + 1) = \text{sum}(a, m) + m + 1.
\]

After we rename the function \(v\) into \(h\) and the parameter \(a\) into \(n\), we get the standard form:

\[
h(n_1, 0) = n_1;
\]

\[
h(n_1, m + 1) = h(n_1, m) + m + 1.
\]

In this standard form, we have \(f(n_1) = n_1\), i.e., \(f = \pi_1\), and \(g(n_1, m, h) = h + m + 1\), i.e., \(g = \text{add}(\pi_3^1, \sigma(\pi_3^2))\).

Let us follow the general scheme for computing primitive recursion. Suppose that we have Turing machines computing the functions \(f(n_1) = n_1\) and \(g(n_1, m, h) = h + m + 1\). Let us show how to build a Turing machine that compute the desired function \(h = PR(f, g)\). We start with the state

\[
\begin{array}{cccc}
- & n_1 & - & x & - & \cdots & \text{start} \\
\end{array}
\]

and we want to end up in the state

\[
\begin{array}{cccc}
- & h(n_1, x) & - & \cdots & \text{halt} \\
\end{array}
\]

This can be done as follows. First, we copy \(x\), add 0, then copy the number \(n_1\), and move the head into the cell right before the second copy of \(n_1\):

\[
\begin{array}{cccc}
- & n_1 & - & x & - & x & - & 0 & - & n_1 & - & \cdots
\end{array}
\]
Then, we apply the Turing machine \( f \). Since a Turing machine never goes beyond the cell where it starts, it will compute the value 

\[ h(n_1, 0) = f(n_1) = n_1, \]

so we will have the following state of the tape:

\[ \begin{array}{ccccccc}
\_ & n_1 & \_ & x & \_ & x & \_ & 0 & \_ & h(n_1, 0) & \_ & \ldots
\end{array} \]

Now, we copy \( n_1 \) and 0 before \( h \), and get

\[ \begin{array}{ccccccc}
\_ & n_1 & \_ & x & \_ & x & \_ & 0 & \_ & n_1 & \_ & 0 & \_ & h(n_1, 0) & \_ & \ldots
\end{array} \]

Then, we apply the Turing machine for computing the function \( g \), and get \( h(n_1, 1) = g(n_1, 0, h(n_1, 0)) \). So, the tape has the form:

\[ \begin{array}{ccccccc}
\_ & n_1 & \_ & x & \_ & x & \_ & 0 & \_ & h(n_1, 1) & \_ & \ldots
\end{array} \]

After that, we decrease the second copy of \( x \) by 1, increase 0 by 1, and get the following:

\[ \begin{array}{ccccccc}
\_ & n_1 & \_ & x & \_ & x & \_ & 0 & \_ & 1 & \_ & h(n_1, 1) & \_ & \ldots
\end{array} \]

and we repeat a similar procedure.

In general, for each \( m \leq x \), we get the following state of the tape:

\[ \begin{array}{ccccccc}
\_ & n_1 & \_ & x & \_ & x & \_ & m & \_ & m & \_ & h(n_1, m) & \_ & \ldots
\end{array} \]

Then, we copy \( n_1 \) and \( m \) before \( h \), and get

\[ \begin{array}{ccccccc}
\_ & n_1 & \_ & n_2 & \_ & x & \_ & x & \_ & m & \_ & m & \_ & h(n_1, m) & \_ & \ldots
\end{array} \]

Now, we apply the Turing machine for computing the function \( g \), and get 

\[ h(n_1, m + 1) = g(n_1, m, h(n_1, m)). \]

So, the tape has the form:

\[ \begin{array}{ccccccc}
\_ & n_1 & \_ & n_2 & \_ & x & \_ & x & \_ & m & \_ & m & \_ & h(n_1, m + 1) & \_ & \ldots
\end{array} \]

Then, we check whether \( x - m = 0 \). If \( x - m > 0 \), we decrease \( x - m \) by 1, increase \( m \) by 1, and get the following:
and we repeat a similar procedure.

Once we get \( x - m = 0 \) i.e., \( m = x \), the state of the tape takes the form

\[
\begin{array}{cccccccc}
- & n_1 & - & x & - & x - (m + 1) & - & m + 1 & - & h(n_1, m + 1) & - & \ldots
\end{array}
\]

Here, we have \( k + 4 = 5 \) numbers:

- the number \( n_1 \), and
- four numbers \( x, 0, x, \) and \( h(n_1, n_2, x) \).

The desired value \( h(n_1, x) \) is 5-th out of 5, so we can get it by applying the Turing machine computing the corresponding projection \( \pi_5 \):

\[
\begin{array}{cccccccc}
\vdash h(n_1, x) & - & \ldots & \text{halt}
\end{array}
\]

This is exactly what we wanted.

In this construction, we use composition, adding 1, subtracting 1, copying, and projection. We know how to do all this on a Turing machine, so indeed we can thus build a Turing machine for computing the function \( PR(f, g) \).