(Ackermann) Function \( \text{ack} \) of two natural variables is defined as follows.

\[
\begin{align*}
\text{ack} \ 0 \ 0 &= 2 \\
\text{ack} \ 1 \ 0 &= 0 \\
\text{ack} \ (m+2) \ 0 &= 1 \\
\text{ack} \ 0 \ (n+1) &= \text{ack} \ 0 \ n + 1 \\
\text{ack} \ (m+1) \ (n+1) &= \text{ack} \ m \ (\text{ack} \ (m+1) \ n)
\end{align*}
\]

(a) Suppose that functions and function application are not implemented expressions; in that case \( n := \text{ack} \ m \ n \) is not a program. Refine \( n := \text{ack} \ m \ n \) to obtain a program.

\[
\begin{align*}
n &= \text{ack} \ m \ n \iff \\
&\quad \text{if } m = n = 0 \text{ then } n := 2 \\
&\quad \text{else if } m = 1 \land n = 0 \text{ then } n := 0 \\
&\quad \text{else if } n = 0 \text{ then } n := 1 \\
&\quad \text{else if } m = 0 \text{ then } n := n - 1. \quad n := \text{ack} \ m \ n. \quad n := n + 1 \\
&\quad \text{else } n := n - 1. \quad n := \text{ack} \ m \ n. \quad m := m - 1. \quad n := \text{ack} \ m \ n. \quad m := m + 1
\end{align*}
\]

Here are the first few values of this function.

<table>
<thead>
<tr>
<th>( m )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>2+n</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>16</td>
<td>65536</td>
<td>*</td>
</tr>
</tbody>
</table>

The entry marked * has about 20000 digits in it, and tower \( n \) means “two to the power two to the power two to the power ...” with \( n \) “two”s. Here is another way to create the table. The top row is 2 3 4 5 and so on; the left column is 2 0 1 1 1 1 and so on; to find an interior item, look left one place, and that's the column number, one row up, to copy from. Just copying; no arithmetic. For example, suppose we want to determine the value of \( \text{ack} \ 3 \ 3 \). Look to the left of position 3 3 and you see 4. So look in the previous row (row 2) under column 4, and you see 16. So \( \text{ack} \ 3 \ 3 = 16 \).

(b) Find a time bound. Hint: you may use function \( \text{ack} \) in your time bound.

\[
\begin{align*}
t' &\leq t + fm n \land n' = \text{ack} \ m \ n \land m'=m \iff \\
&\quad \text{if } m = n = 0 \text{ then } n := 2 \\
&\quad \text{else if } m = 1 \land n = 0 \text{ then } n := 0 \\
&\quad \text{else if } n = 0 \text{ then } n := 1 \\
&\quad \text{else if } m = 0 \text{ then } \\
&\quad \quad \text{then } n := n - 1. \quad t := t + 1. \quad t' \leq t + fm n \land n' = \text{ack} \ m \ n \land m'=m. \quad n := n + 1 \\
&\quad \quad \text{else } n := n - 1. \quad t := t + 1. \quad t' \leq t + fm n \land n' = \text{ack} \ m \ n \land m'=m. \quad m := m - 1. \quad t' \leq t + fm n \land n' = \text{ack} \ m \ n \land m'=m. \quad m := m + 1
\end{align*}
\]

In the last alternative, I put \( t := t + 1 \) before the first recursive call, but not before the second. The one occurrence ensures that every loop includes a time increment. But I could have put another one in. Using Refinement by Cases, and throwing away the unnecessary pieces, we need \( f \) to satisfy five things.

\[
\begin{align*}
t' &\leq t + fm n \iff m = n = 0 \land t = t \\
t' &\leq t + fm n \iff m = 1 \land n = 0 \land t = t \\
t' &\leq t + fm n \iff m > 1 \land n = 0 \land t = t \\
t' &\leq t + fm n \iff m = 0 \land n > 0 \land t' \leq t + 1 + fm \ (n-1)
\end{align*}
\]
\[ t' \leq t + f m n \iff m \geq 0 \land n \geq 0 \land t' \leq t + 1 + f m (n-1) + f (m-1) (ack m (n-1)) \]

Simplifying,
\[
\begin{align*}
  f_0 0 & \geq 0 \\
  f 0 (n+1) & \geq f 0 n + 1 \\
  f (m+1) (n+1) & \geq f (m+1) n + f m (ack (m+1) n) + 1
\end{align*}
\]

These are the constraints on \( f \). So replace \( \geq \) by \( = \) and we have a definition of \( f \) that gives the exact execution time (in terms of \( \text{ack} \)).

(c) Find a space bound.