1. (a)

Here is the AST, with one node on each numbered line (arbitrarily numbered breadth-first).

```
1: let f
    / \
   /    \
2: fun x
   |     \
3:   fun y
4:    x  \
5:      |
6:      if
    /    \
7:     y  \
8:      z  \
9:       @
    /    \
   /    \
10:  f   \
11:    3
```

From this tree, we generate the following constraints:

<table>
<thead>
<tr>
<th>Node #</th>
<th>Rule</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>let</td>
<td>t2 = tf and t1 = t3</td>
</tr>
<tr>
<td>2</td>
<td>fun</td>
<td>t2 = tx -&gt; t4</td>
</tr>
<tr>
<td>3</td>
<td>fun</td>
<td>t3 = ty -&gt; t5</td>
</tr>
<tr>
<td>4</td>
<td>var</td>
<td>t4 = tx</td>
</tr>
<tr>
<td>5</td>
<td>fun</td>
<td>t5 = tz -&gt; t6</td>
</tr>
<tr>
<td>6</td>
<td>if</td>
<td>t7 = Bool and t6 = t8 = t9</td>
</tr>
<tr>
<td>7</td>
<td>var</td>
<td>t7 = ty</td>
</tr>
<tr>
<td>8</td>
<td>var</td>
<td>t8 = tz</td>
</tr>
<tr>
<td>9</td>
<td>app</td>
<td>t10 = t11 -&gt; t9</td>
</tr>
<tr>
<td>10</td>
<td>var</td>
<td>t10 = tf</td>
</tr>
<tr>
<td>11</td>
<td>int</td>
<td>t11 = Int</td>
</tr>
</tbody>
</table>

We can solve this by inspection:

First, using the identities for t2, t4, t7, t8, t10, t11 we can substitute for these variables, leading to the following modified constraints:

```
2'      tf = tx -> tx
6'      ty = Bool and t6 = tz = t9
9'      tf = Int -> t9
```

Using t1 = t3, we can substitute for t3 to get the modified constraint

```
3'     t1 = ty -> t5
```
(Choosing whether to get rid of \( t_1 \) or \( t_3 \) is fairly arbitrary, but we ultimately want to know the root expression type \( t_1 \), so we keep that.)

Similarly, from \( t_6 = t_9 \), we can substitute for \( t_6 \) and \( t_9 \) (again fairly arbitrary, but we ultimately want to know \( t_2 \)), getting

\[
\begin{align*}
5' & \quad t_5 = t_2 \to t_2 \\
9' & \quad t_f = \text{Int} \to t_2 \\
\end{align*}
\]

Equating the two expressions (2’ and 9”) for \( t_f \), we get that

\[ t_x \to t_x = \text{Int} \to t_2 \]

which implies that \( t_x = t_2 = \text{Int} \).

Summarizing, we have

\[
\begin{align*}
t_1 &= \text{Bool} \to (\text{Int} \to \text{Int}) \\
t_f &= \text{Int} \to \text{Int} \\
t_x &= \text{Int} \\
t_y &= \text{Bool} \\
t_z &= \text{Int} \\
\end{align*}
\]

(b)

The AST:

1: fun f \\
| \\
2: fun g \\
| \\
3: fun x \\
| \\
4: @ \\
/ \ \\
5: f \ \\
6: @ \\
/ \ \\
7: g \ \\
8: x

The generated constraints:

<table>
<thead>
<tr>
<th>Node #</th>
<th>Rule</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>fun</td>
<td>( t_1 = t_f \to t_2 )</td>
</tr>
<tr>
<td>2</td>
<td>fun</td>
<td>( t_2 = t_g \to t_3 )</td>
</tr>
<tr>
<td>3</td>
<td>fun</td>
<td>( t_3 = t_x \to t_4 )</td>
</tr>
<tr>
<td>4</td>
<td>app</td>
<td>( t_5 = t_6 \to t_4 )</td>
</tr>
</tbody>
</table>
Solution by inspection:

Using the identities from nodes 5, 6, and 7, we can rewrite the constraints for nodes 4 and 6 as

\[
\begin{align*}
4' & \quad tf = t6 \rightarrow t4 \\
6' & \quad tg = tx \rightarrow t6
\end{align*}
\]

There are no other constraints on \(tx\), \(t4\), and \(t6\), so the overall type must be parametric in (i.e. polymorphic over) these types. The type of the overall expression is

\[
\begin{align*}
t1 &= tf \rightarrow t2 \quad \text{(by 1)} \\
    &= tf \rightarrow (tg \rightarrow t3) \quad \text{(by 2)} \\
    &= tf \rightarrow (tg \rightarrow (tx \rightarrow t4)) \quad \text{(by 3)} \\
    &= (t6 \rightarrow t4) \rightarrow ((tx \rightarrow t6) \rightarrow (tx \rightarrow t4)) \quad \text{(by 4' and 6')} \\
\end{align*}
\]

Or, using more suggesting names for the polymorphic types, and the convention that \(\rightarrow\) associates to the right:

\[
\begin{align*}
t1 &= (tb \rightarrow tc) \rightarrow (ta \rightarrow tb) \rightarrow (ta \rightarrow tc) \\
tx &= ta \\
tf &= tb \rightarrow tc \\
tg &= ta \rightarrow tb
\end{align*}
\]

which makes sense for a general-purpose “compose” function.