$$
\begin{array}{ll}
 & \texttt{factorial}\ c_3 \\
= & \texttt{fix g}\ c_3 \\
\longrightarrow & \texttt{h h}\ c_3 \\
 & \text{where } \texttt{h} = \lambda\texttt{x. g } (\lambda\texttt{y. x x y}) \\
\longrightarrow & \texttt{g fct}\ c_3 \\
 & \text{where } \texttt{fct} = \lambda\texttt{y. h h y} \\
\longrightarrow & (\lambda\texttt{n. if realeq n}\ c_0 \\
 & \qquad\quad \texttt{then}\ c_1 \\
 & \qquad\quad \texttt{else times n (fct (prd n)))} \\
 & \qquad c_3 \\
\longrightarrow & \texttt{if realeq}\ c_3\ c_0 \\
 & \quad \texttt{then}\ c_1 \\
 & \quad \texttt{else times}\ c_3\ \texttt{(fct (prd}\ c_3\texttt{))} \\
\longrightarrow^* & \texttt{times}\ c_3\ \texttt{(fct (prd}\ c_3\texttt{))} \\
\longrightarrow^* & \texttt{times}\ c_3\ \texttt{(fct}\ c_2'\texttt{)} \\
 & \text{where } c_2' \text{ is behaviorally equivalent to } c_2 \\
\longrightarrow^* & \texttt{times}\ c_3\ \texttt{(g fct}\ c_2'\texttt{)} \\
\longrightarrow^* & \texttt{times}\ c_3\ \texttt{(times}\ c_2'\ \texttt{(g fct}\ c_1'\texttt{)).} \\
 & \text{where } c_1' \text{ is behaviorally equivalent to } c_1 \\
 & \text{(by repeating the same calculation for } \texttt{g fct}\ c_2'\text{)} \\
\longrightarrow^* & \texttt{times}\ c_3\ \texttt{(times}\ c_2'\ \texttt{(times}\ c_1'\ \texttt{(g fct}\ c_0'\texttt{))).} \\
 & \text{where } c_0' \text{ is behaviorally equivalent to } c_0 \\
 & \text{(similarly)} \\
\longrightarrow^* & \texttt{times}\ c_3\ \texttt{(times}\ c_2'\ \texttt{(times}\ c_1'\ \texttt{(if realeq}\ c_0'\ c_0\ \texttt{then}\ c_1 \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{else ...)))} \\
\longrightarrow^* & \texttt{times}\ c_3\ \texttt{(times}\ c_2'\ \texttt{(times}\ c_1'\ c_1\texttt{))} \\
\longrightarrow^* & c_6' \\
 & \text{where } c_6' \text{ is behaviorally equivalent to } c_6.
\end{array}
$$

Figure 5-2: Evaluation of `factorial` c_3

Representation

Before leaving our examples behind and proceeding to the formal definition of the lambda-calculus, we should pause for one final question: What, exactly, does it mean to say that the Church numerals *represent* ordinary numbers?

To answer, we first need to remind ourselves of what the ordinary numbers are. There are many (equivalent) ways to define them; the one we have chosen here (in Figure 3-2) is to give:

- a constant 0,

$$\frac{}{i \Downarrow i}\ \textsc{Int}$$

$$\frac{e_1 \Downarrow i_1 \qquad e_2 \Downarrow i_2 \qquad \oplus \in \{+,-,*\} \qquad (i_3 = i_1 \oplus i_2)}{e_1 \oplus e_2 \Downarrow i_3}\ \textsc{Arith}$$

$$\frac{e_1 \Downarrow i_1 \qquad e_2 \Downarrow i_2 \qquad (i_1 \odot i_2 \text{ holds})}{e_1 \odot e_2 \Downarrow \mathsf{true}}\ \textsc{PredTrue}$$

$$\frac{e_1 \Downarrow i_1 \qquad e_2 \Downarrow i_2 \qquad (i_1 \odot i_2 \text{ does not hold})}{e_1 \odot e_2 \Downarrow \mathsf{false}}\ \textsc{PredFalse}$$

$$\frac{e_1 \Downarrow \mathsf{true} \qquad e_2 \Downarrow v}{\mathsf{if}\, e_1\, \mathsf{then}\, e_2\, \mathsf{else}\, e_3 \Downarrow v}\ \textsc{IfTrue} \quad \frac{e_1 \Downarrow \mathsf{false} \qquad e_3 \Downarrow v}{\mathsf{if}\, e_1\, \mathsf{then}\, e_2\, \mathsf{else}\, e_3 \Downarrow v}\ \textsc{IfFalse}$$

$$\frac{}{\mathsf{lambda}\, x.\ e \Downarrow \mathsf{lambda}\, x.\ e}\ \textsc{Lambda}$$

$$\frac{e_1 \Downarrow \mathsf{lambda}\, x.\ e_1' \qquad e_2 \Downarrow v \qquad e_1'[x \mapsto v] \Downarrow v'}{(e_1\ e_2) \Downarrow v'}\ \textsc{App}$$

$$\frac{e_1 \Downarrow v_1 \qquad e_2[x \mapsto v_1] \Downarrow v_2}{\mathsf{let}\, x = e_1\, \mathsf{in}\, e_2 \Downarrow v_2}\ \textsc{Let} \qquad \frac{e[f \mapsto \mathsf{fix}\, f\ \mathsf{is}\ e] \Downarrow v}{\mathsf{fix}\, f\ \mathsf{is}\ e \Downarrow v}\ \textsc{Fix}$$

$$\frac{}{\mathsf{Nil} \Downarrow \mathsf{Nil}}\ \textsc{Nil} \qquad \frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{e_1 :: e_2 \Downarrow v_1 :: v_2}\ \textsc{Cons}$$

$$\frac{e_1 \Downarrow \mathsf{Nil} \qquad e_2 \Downarrow v}{\mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{Nil} \to e_2 \mid x :: y \to e_3\ \mathsf{end} \Downarrow v}\ \textsc{MatchNil}$$

$$\frac{e_1 \Downarrow v_1 :: v_2 \qquad e_3[x \mapsto v_1][y \mapsto v_2] \Downarrow v_3}{\mathsf{match}\ e_1\ \mathsf{with}\ \mathsf{Nil} \to e_2 \mid x :: y \to e_3\ \mathsf{end} \Downarrow v_3}\ \textsc{MatchCons}$$

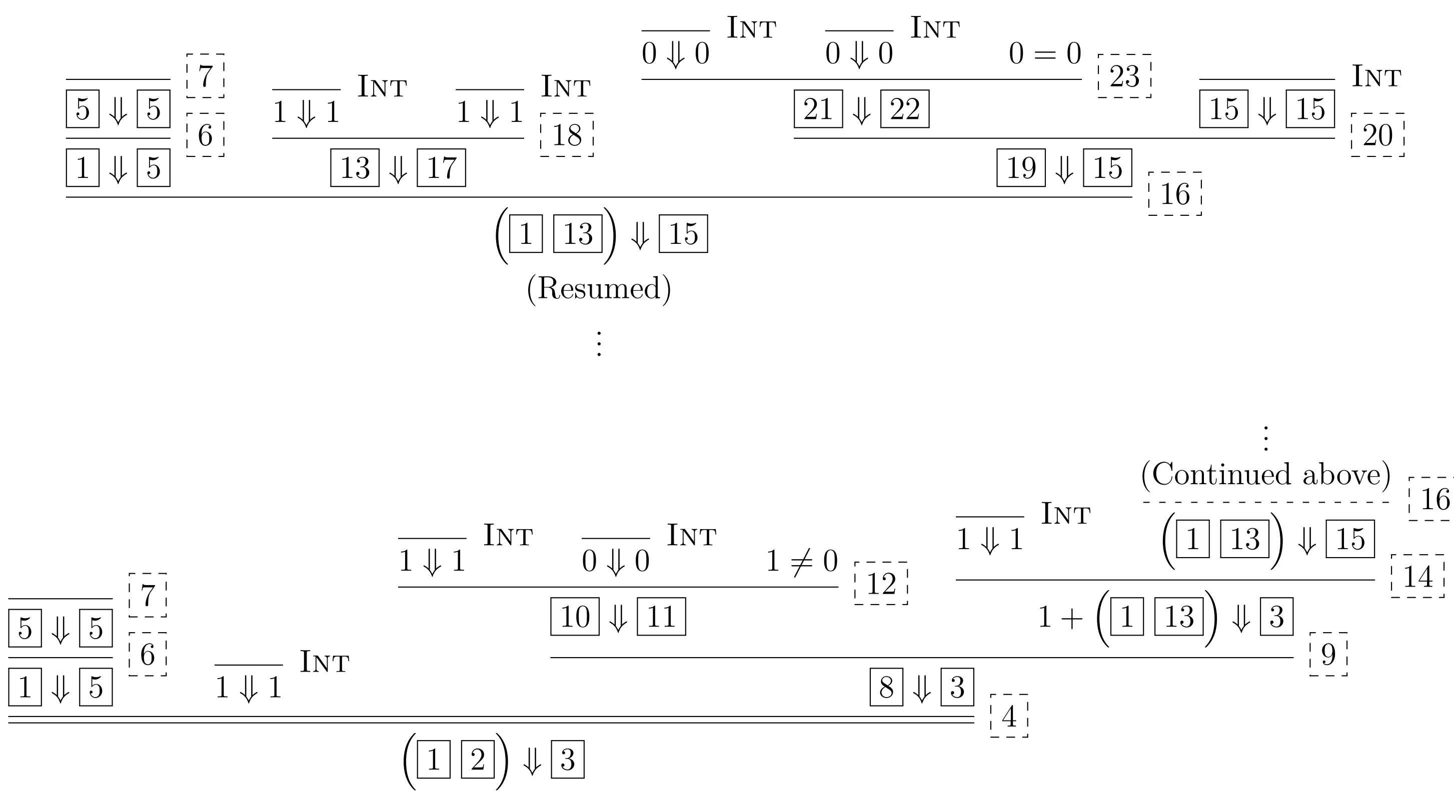
Problem 4 (20 points)

Consider the following (untyped) λ^+ expression:

```
(fix r is lambda n.
    if n = 0 then 5 else n + r (n - 1)) 1
```

Denote the expression above as e. Below is the skeleton of the derivation tree that shows $e \Downarrow v$ for some v. The derivation starts from the bottom of the page with the double bar, and proceeds upwards.

Complete the derivation tree by filling in the table on the next page. A $\boxed{\text{solid box}}$ should be filled with an λ^+ expression, while a dashed box should be filled with the name of an *operational semantics* rule.



Always write the full, concrete expression or rule name in a cell. Do *not* use abbreviations, and do *not* use names ("*e*", "*v*", etc.) or numbers to refer to other expressions or rules.

#	Expression/Rule Name
1	`(fix r is lambda n. if n = 0 then 5 else n + r (n - 1))`
2	`1`
3	(JR) 6
4	App
5	`(lambda n. if n = 0 then 5 else n + (fix r is lambda n. if n = 0 then 5 else n + r (n - 1))(n - 1))`
6	(JR) Fix
7	(JR) Lambda
8	`if 1 = 0 then 5 else 1 + (fix r is lambda n. if n = 0 then 5 else n + r (n - 1))(1 - 1))`
9	(JR) IfFalse
10	(JR) 1=0
11	(JR) false

#	Expression/Rule Name
12	(JR) PredFalse
13	(JR) 1-1
14	(JR) Arith
15	(JR) 5
16	(JR) App
17	(JR) 0
18	(JR) Arith
19	`if 0 = 0 then 5 else 0 + (fix r is lambda n. if n = 0 then 5 else n + r (n - 1))(0 - 1))`
20	(JR) IfTrue
21	(JR) 0=0
22	(JR) true
23	(JR) PredTrue