Problem 1

a)

$$\dfrac{\dfrac{\dfrac{f \in domain(s')}{\langle f, s'\rangle \Downarrow_e \langle n_1, s'\rangle} \qquad \dfrac{c \in domain(s')}{\langle c, s'\rangle \Downarrow_e \langle n_2, s'\rangle}}{\langle f \times c, s'\rangle \Downarrow_e \langle n_1 \times n_2, s'\rangle}}{\langle f := f \times c, s'\rangle \Downarrow_c \langle s'[f \text{ -> } n_1 \times n_2]\rangle}$$

b)

i)

x is assigned the value of E_1.
When x is less than E_2 or equal to, the command C gets executed.
Then the command is repeated with E1 set to the value of x+1, until x is greater than E_2.
The for loop can become infinite when we have the same variable x in E_2, e.g.

```
for x from 1 to x do skip
```

This command will never be able to terminate.

ii)

This part is omitted, but note that the tree is pretty big, and error-prone to draw.

Final value of f = 6 and c = 4

c)

i)

Incorrect:

$$\frac{\langle E\_1, s\rangle \text{ -> } \langle E\_1', s'\rangle}{\langle \text{for x from E\_1 to E\_2 do C}, s\rangle \text{ -> } \langle \text{for x from E\_1' to E\_2 do C}, s'\rangle}$$

$$\frac{\langle E\_2, s\rangle \text{ -> } \langle E\_2', s'\rangle}{\langle \text{for x from n\_1 to E\_2 do C}, s\rangle \text{ -> } \langle \text{for x from n\_1 to E\_2' do C}, s'\rangle}$$

$$\frac{\langle n\_1 \text{ <= } n\_2, s\rangle \text{ -> } \langle \text{true}, s\rangle}{\langle \text{for x from n\_1 to n\_2 do C}, s\rangle \text{ -> } \langle \text{x := n\_1; C; for x from x+1 to n\_2 do C}, s\rangle}$$

$$\frac{\langle n\_1 \text{ <= } n\_2, s\rangle \text{ -> } \langle \text{false}, s\rangle}{\langle \text{for x from n\_1 to n\_2 do C}, s\rangle \text{ -> } \langle s\rangle}$$

See slide 18-19 of Lecture 2, the proper way is to unfold.

Slide 18

Incorrect Semantics for `while`

(W-WHILE?)
$$\frac{\langle B, s\rangle \rightarrow_b \langle B', s'\rangle}{\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \rightarrow_c \langle \texttt{while}\ B'\ \texttt{do}\ C, s'\rangle}$$

(W-WHILE?)
$$\frac{}{\langle \texttt{while}\ \texttt{false}\ \texttt{do}\ C, s\rangle \rightarrow_c \langle \texttt{skip}, s\rangle}$$

(W-WHILE?)
$$\frac{}{\langle \texttt{while}\ \texttt{true}\ \texttt{do}\ C, s\rangle \rightarrow_c \langle ? \rangle}$$

Slide 19

Correct Semantics for `while`

(W-WHILE)
$$\frac{}{\begin{array}{l}\langle \texttt{while}\ B\ \texttt{do}\ C, s\rangle \rightarrow_c \\ \langle \texttt{if}\ B\ \texttt{then}\ (C; \texttt{while}\ B\ \texttt{do}\ C)\ \text{else}\ \texttt{skip}, s\rangle\end{array}}$$

All this rule does is 'unfold' the while loop once. If we could write down the infinite unfolding, there would be no need for the `while` syntax.

Alternative solution, similar to definition of while:
<for x from E1 to E2 do C, s> ->
<x := E1; if x > E2 skip else (C; for x from x+1 to E2 do C), s>

ii)
This part is omitted.
Would we have to write <skip; C, s> -> <C, s> for everything? Makes it quite a bit longer

Problem 2

a)

Computable functions

Definition. The partial function $f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ is **(register machine) computable** if there is a register machine M with at least $n+1$ registers $R_0, R_1, \dots, R_n$ (and maybe more) such that for all $(x_1, \dots, x_n) \in \mathbb{N}^n$ and all $y \in \mathbb{N}$,

> the computation of M starting with $R_0 = 0$, $R_1 = x_1, \dots,$ $R_n = x_n$ and all other registers set to 0, halts with $R_0 = y$

if and only if $f(x_1, \dots, x_n) = y$.

i)

ii)

L_0: R_1- -> L_1, L_1

L_1: HALT

iii)

The input is a number n. The output is n = <x, y> = 2^x(2y + 1) - 1 where R_0 = y

The first instruction means the remaining problem is n+1 = <<x, y>> = 2^x(2y+1).

Also R3 = x

b)

i and ii are pretty standard

If anyone has an answer to this that would be great =)

b) i) Second (pair m n) $\rightarrow$ n.

Second (pair m n) $= \lambda p. p (\lambda x. \lambda y. y)$ pair m n

$=$ pair m n $(\lambda x. \lambda y. y)$

$= (\lambda x \lambda y \lambda f. f x y)\ \underline{m}\ \underline{n}\ \underline{(\lambda x. \lambda y. y)}$

$= \lambda x. \lambda y. y\ m\ n.$

$= n =$ RHS // (SHOWN)

ii). IF succ(n) $\rightarrow$ n+1

Shift-Inc $\triangleq \lambda x.$ pair(second x) (succ (second x))

SHOW

Shift-Inc (pair m n) = (pair n n+1)

LHS: Shift-Inc (pair m n)

$= \lambda x$ (pair second x) (succ(second x)) (pair m n)

= (pair(second (pair m n))) (succ (second (pair m n)))

From b) i), replace both second(pair m n) with n.

= (pair n) (succ n)

= pair n succ n.

= pair n n+1 = RHS // (SHOWN)

iii) λp. first(p shift-inc (pair 0 0))

p will be of the form λf. λx. f^n x (which represents n)
shift-inc is passed in as f
pair 0 0 is passed in as x
shift-inc is therefore applied n times to pair 0 0 which leaves pair n-1 n
first (pair n-1 n) reduces to n-1