cPlease critique and improve-*

1.

a.

i) $(\text{while } x+1 \text{ do } x := x+1,\ x \mapsto 0)$

$$\cfrac{\cfrac{\cfrac{s(x)=0}{(x,s)\Downarrow_e 0} \qquad \cfrac{1 \in \mathbb{N}}{\langle 1,s\rangle \Downarrow_e 1} \qquad 0 \neq 1}{(x \neq 1, s) \Downarrow_b \text{true}} \qquad [C] \qquad [W]}{(\text{while } (x+1) \text{ do } (x := x+1), s) \Downarrow_c s[x \mapsto 1]}$$

$$[C]\quad \cfrac{\cfrac{\cfrac{s(x)=0}{\langle x,s\rangle \Downarrow 0} \qquad \cfrac{1 \in \mathbb{N}}{\langle 1,s\rangle \Downarrow 1} \qquad 0+1=1}{\langle x+1,s\rangle \Downarrow 1}}{\langle x := x+1,\ s\rangle \Downarrow s[x \mapsto 1]}$$

$$[W]\quad \cfrac{\cfrac{\cfrac{s(x)=1}{\langle x,s\rangle \Downarrow 1} \qquad \cfrac{1 \in \mathbb{N}}{\langle 1,s\rangle \Downarrow 1} \qquad 1=1}{\langle x \neq 1, x \mapsto 1\rangle \Downarrow_b \text{false}}}{\langle \text{while } (x \neq 1) \text{ do } (x := x+1), (x \mapsto 1)\rangle \Downarrow_c s[x \mapsto 1]}$$

i.

ii. ($\Downarrow_c$ is total if forall there E exists a v)

$\Downarrow_c$ is not total because we can use a variable that is not defined, i.e. $s(x) = n$is never true. The other way in which $\Downarrow_c$ is not total is for an expression such as: while true do x := x + 1, as this would never finish.

(I think it's also important to mention the composition of commands might also lead to a program not terminating, so there's 3 ways in which $\Downarrow_c$is not total)

Alternative answer:

All the command rules only modify the state without actually reducing to a value.

b.

i. $P(E) = \exists v. E \Downarrow_{e'} v$

<u>Base Cases</u>

$$P(n) = \exists v. n \Downarrow_{e'} v$$

Holds by setting $v = n$, then holds by $\frac{n \in N}{<n,s> \Downarrow_{e'} n}$.

$$P(x) = \exists v. x \Downarrow_{e'} v$$

Two cases to consider, $x \in dom(s)$,

Apply $\frac{s(x)=n}{<x,s> \Downarrow_{e'} n}$. So set $v = n$, holds.

$x \notin dom(s)$,

Apply $\frac{x \notin dom(s)}{<x,s>\Downarrow_{e'} error}$. So set $v = error$, holds.

<u>Inductive step</u>

$$P(E_1 + E_2) = \exists v_3. (E_1 + E_2) \Downarrow_{e'} v_3$$

Inductive hypotheses:

Assume $\exists v_1. E_1 \Downarrow_{e'} v_1$ and $\exists v_2. E_2 \Downarrow_{e'} v_2$ hold.

Only one rule applies. By this rule (can't be bothered to write it out), since we have $v_1, v_2$, and $E_1 + E_2 \Downarrow_{e'} v_3$ where $v_3 = v_1 + v_2$ exists (either as a natural number or error), so holds.

ii. Assuming we already have $\Downarrow_{e'}$ we propagate *error* forward.

Imagine all the existing rules and add:

$$\frac{<E,s>\Downarrow_{e'} error}{<x:=E,s>\Downarrow_{c'} error} \qquad \frac{<C_1,s>\Downarrow_{c'} error}{<C_1;C_2,s>\Downarrow_{c'} error} \quad \frac{<C_1,s>\Downarrow_{c'} s' \qquad <C_2,s'>\Downarrow_{c'} error}{<C_1;C_2,s>\Downarrow_{c'} error}$$

+ while rules which also have error propagating 'left to right'.

iii. $\Downarrow_{c'}$is still not total because we can create situations where there is an infinite loop.
e.g. <while true do x:=x+1, x->0>

We could adjust c' to catch this example, but can always reformulate the condition or loop body to get around this. Hence c' is not total.

2.

a.

i. ~~Since the function is f: NxN -> N I'm guessing that $R_0$ and $R_1$ are the argument registers and that $R_0$is the result register.~~
~~With this assumption, the function computed is $R_0 = R_0 + R_1$, i.e. addition.~~

By definition on the slides, this might be a trick question.
$R_0$ is 0 by default, and contains the value that f computes once the machine halts. Then $R_1$, $R_2$ correspond to the two arguments. The machine does not make use of $R_2$, but that doesn't matter.

f returns the first of its arguments, i.e. copies $R_1$into $R_0$.

$$f: \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$f(x, y) = x$

ii.

L0 = R1- → L1, L2 = ⟪3, ⟨1,2⟩⟫ = 152
L1 = R0+ → L0 = ⟪0, 0⟫ = 1
L2 = HALT = 0

Prog = list(152, 1, 0) = ⟪152, ⟪1, ⟪0, **0**⟫⟫⟫ = ⟪152, ⟪1, 1⟫⟫ = ⟪152, 6⟫ = $2^{152} \times 13$

iii. R_0 is the result, 0 by default
R_1 is the program (same as above)
R_2 is the list of arguments ⟪2,⟪1, 0⟫⟫=⟪2, 2⟫= $2^2 \times 5$.

All other registers are set to 0.

b.
~~i. I found it more intuitive to convert the rules into actual lambda calculus~~

~~To show (SKK)x = Ix~~

~~(SKK)x =~~ $\cancel{[(\lambda xyz. xz(yz))\ K\ K]\ x}$ ~~by RED-S~~

$\rightarrow_\beta \cancel{[(\lambda yz. Kz(yz))\ K]\ x}$

$\rightarrow_\beta \cancel{(\lambda z. Kz(Kz))\ x}$

$\cancel{\mathrm{K}z = (\lambda xy. x)z \rightarrow_\beta \lambda y. z}$

~~So~~ $\cancel{[\lambda z. (\lambda xy. x)z(\lambda y. z)]x \rightarrow_\beta [\lambda z. (\lambda y. z)(\lambda y. z)]x \rightarrow_\beta [\lambda z. z]x}$

$\cancel{[\lambda z. z]x}$~~= Ix from RED-I~~

*Alternatively, using their rules:*

b) i. SKK) x -> Kx (Kx)
-> x

ii. (SII) x -> Ix (Ix)
-> x (Ix)
-> x x

iii. S' S' = (S (Kx) (SII)) S'
-> (Kx) S' ((SII) S')
-> x ((SII) S')
-> x (IS' (IS'))
-> x (S' (IS'))
-> x (S' S')

iii. *Alternative: Last 3 steps can be replaced with single step using rule from part ii*
S' S' = (S (Kx) (SII)) S'
-> (Kx) S' ((SII) S')
-> x ((SII) S')
**-> x (S' S')**

Let P(n) ≡ (SII) S' = $x^n$ (S' S'). We will show that for all n ∈ N+.P(n) holds by induction over n.

*Base case:*

We must show that P(1) holds. We have that:
(SII) S' -> IS' (IS')
-> S' (IS')
-> S' S'
-> x (S' S') by previous part

$= x^1$ (S' S') by definition provided

*Inductive case:*

We must show that P(k+1) holds, assuming P(k) ≡ (SII) S' = $x^k$ (S' S') (not sure how we can really use this) as our inductive hypothesis. We have that:

(SII) S'-> IS' (IS')
-> S' (IS')
-> S' S'
-> x (S' S')
-> x (x (S' S'))
-*> x ( … x (x (S' S')))
(k times)
-> x ( … x ($x^2$ (S' S')))
(k - 1 times)
-> x ( … x ($x^3$ (S' S')))
(k - 2 times)
-*> x ($x^k$ (S' S'))
= $x^{k+1}$ (S' S') by definition provided

***

Proving P(k+1) using P(k):

Could apply the provided definition of equality - show that $x^{k+1}$ (S' S') →* z and (SII)S' →* z for some common z.

$x^{k+1}$ (S' S') →* x($x^k$ (S' S')) by def
→* x((SII) S') by inductive hypothesis P(k)
→* x(S'S') by Q2(b)(ii)
→* S'S' by first part of this question (show S'S' = x(S'S'))

(SII)S' →* S'S' by Q2(b)(ii)

By definition of the equality relation between combinator terms provided on the question paper, $x^{k+1}$ (S' S') = (SII) S', as required.

***