3.
Might be completely wrong cos I did this hella hungover but it’s better than no answers eh?

a) s = (x -> 2, y -> 3)
i) <(z + x) + (y + 4), s>

--------------------------------- $z \notin dom(s)$

<z, s> $\to_e$ fault

<z + x, s> $\to_e$ fault

<(z + x) + (y + 4), s> -->$_e$ fault

<(1 + x) + (y + z), s>

-------------------------------------- s(x) = 2

<x, s> $\to_e$ <2, s>

<1 + x, s> $\to_e$ <1+2, s>

<(1 + x) + (y + z), s> $\to_e$ <(1+2) + (y + z), s>

<(1 + x) + (y + 4), s>

-------------------------------------- s(x) = 2

<x, s> $\to_e$ <2, s>

<1 + x, s> $\to_e$ <(1+2), s>

<(1 + x) + (y + 4), s> $\to_e$ <(1 +2)+ (y + 4), s>

ii)

<(z + x) + (y + 4), s> → fault

<(1 + x) + (y + z), s> → <(1 + 2) + (y + z), s>

→ <3 + (y + z), s> → <3 + (3 + z), s> → fault

<(1 + x) + (y + 4), s> → <(1 + 2) + (y + 4), s>

→ <3 + (y + 4), s> → <3 + (3 + 4), s> → <3 + 7, s> → <10, s>

i)
When E = n, then in this case the predicate doesn’t hold, so the implication holds trivially.
When E = x, then it’s easy to see from the rule that we have the expected result.
When E = E1 + E2, then consider three separate cases E = E1 + E2, E = n + E2 and E = n1 + n2.
ii) Just follows the normal mathematical induction rule.
  Base Case: Proved in (i)
  Inductive Case: Break into k steps and 1 final step and then use the IH
iii)
When E = n, then the predicate doesn’t hold, so we have the implication trivially.
When E = x, then we just inspect the rule and get the result.
When E = E1 + E2 then consider two cases, either E = E1 + E2 or E = n + E2
Use the examples given to you in (ai) where it faults
iv) Similar to ii
v) Strong normalization implies that every term has normalised form. Specifically in our case, every expression **must** eventually becomes either a natural number **n** or **fault**. If the given expression reduced to a **fault**, i.e. <E, s> →e* <fault, s>, the from iv’s result we have `var(E) - dom(s) != empty set`. And if the given expression reduced to a natural number **n** from ii it must be the case `var(E) U dom(s) = var(n) U dom(s) = dom(s)`, which means we have `var(E)` is a subset of `dom(s)`
4.
a) There exists a register machine M with at least n + 1 registers, R_0 , R_1 , … , R_n , such that for all (x_1 , ... , x_n) ϵ ℕ^n and all y ϵ ℕ :
  The computation of M starting with R_0 = 0 , R_1 = x_1 , … , R_n = x_n and all other registers set to 0, halts with R_0 = y
  If and only if f(x_1 , ... , x_n) = y
b)
i) Some diagram (hopefully something like the one below)?

```
START ->  R1-  <-  -> R2- -->> HALT
           |                    ^
          V      __________   /
          V   /
          R2-   - ->> R0+ <-
```

ii) f(x, y) computed by M is the function that halts if y != x
c) inc(R): L_0: R^+ -> L_j

zero(R): L_0: R^- -> L_0, L_j

test(R_i, R_j): L_0: R_i^- -> L_1, L_2
L_1: R_j^- -> L_0, L_l
L_2: R_j^- -> L_l, L_k

Define ADD(a,x,y) as: ENTRY → L_0

L_0: R_a^- → L_1, EXIT

L_1: R_x^+ → L_2

L_2: R_y^+ → L_0

Then test(R_i,R_j) is: L_0: R_i^- → L_1, L_3

L_1: R_j^- → L_2, L_7

L_2: R_a^+ → L_0

L_3: R_j^- → L_4, L_6 This section is for x < y

L_4: ADD(a, i, j) → L_5

L_5: R_j^+ → L_l

L_6: ADD(a, i, j) → L_k x = y (success)

L_7: ADD(a, i, j) → L_8 x > y

L_8: R_j^+ → L_l

HALT: HALT

d) i) A register machine is said to be decided the halting problem if for all e,a_1, a_2, a_3, a_n which all natural number, it always halt with R_0 equals to 0 and 1 when starting with R_0 = 0, R1 = e and R_2 = [a_1,a_2,...,a_n]. And R_0 equals to 1 if and only if a register machine executed the program e with initial register value set as R=0, R_1 = a_1, R_2 = a_2, …,R_n = a_n, and all other register zeroed halts. This register machine doesn't exist, so we say that halting problem is undecidable for a register machine.

ii)
We cannot express the halting problem using successor machine because all the operation that is supported by the successor machine can be implemented using register machine. And we can't construct a register machine that is capable of deciding the halting problem, thus we cannot do it using the successor machine either.

Or we could prove by contradiction. Assuming that there is a successor machine **S** which can solve the halting problem. Then by using the result from part c, we will obtain a register machine **M** that is capable of solving the halting problem. However as we have known already such register machine **M** doesn't exist. So we have a contradiction. Thus it must be that we can't have a successor machine **S** which can solve the halting problem.