2021 Models of Computation:

(What a mess) (using mess to describe this is just not apt)

1a)

I] TODO: Upload neater version

i)

$$\dfrac{B\,\dfrac{\langle x+1,(x\mapsto 0)\rangle \Downarrow 1 \qquad s[x\mapsto 1]=s'}{\langle x:=x+1,(x\mapsto 0)\rangle \Downarrow (x\mapsto 1)} \qquad F\,\dfrac{B\,\dfrac{\langle x+1,(x\mapsto 1)\rangle \Downarrow 2 \qquad s'[x\mapsto 2]=s''}{\langle x:=x+1,(x\mapsto 1)\rangle \Downarrow (x\mapsto 2)} \qquad G\,\dfrac{}{\langle loop(x:x+1),s''\rangle \Downarrow s''}}{\langle loop\,(x: x+1),(x\mapsto 1)\rangle \Downarrow s''}}{\langle loop(x:=x+1),(x\mapsto 0)\rangle \Downarrow s''}\,F$$

ii] Explain in words the behaviour of the loop command.

Loop(C) is equivalent to doing C, and doing C again on the modified state. If there are no further changes to the state, we can stop there.

iii] Let <C, s_0> ⇓ s' for some store s', where C and s_0 are defined as in part a)i). What are all the possible values of x in s'? Justify your answer in words.

X can be all the (positive?) integers, (inclusive of 0). (Easier to say natural number)

When x = 0, loop(x:= x+1), s[x->0]> ⇓ s[x->0] using the no-IH derivation rule.

Otherwise, apply the usual loop-decomposition rule. If x = n, apply that rule n times. (I.e. do x:=x+1 n times).

As the integers are countable, we know that n \in N is finite, s'[x->n] can be reached.

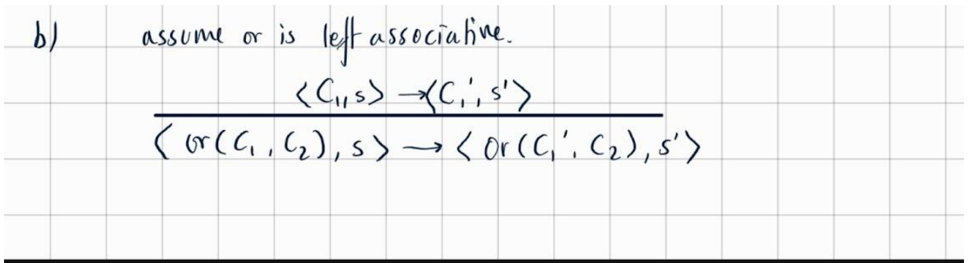
X can be any natural number since we have free choice how many times to execute the loop body before breaking out with the other derivation rule. We can even immediately break the loop after 0 iterations, and so 0 is also a valid x value.

B) Give the small-step operational semantics rules for or(C1, C2) and loop(C).

C1->C1'   C2 -> C2'

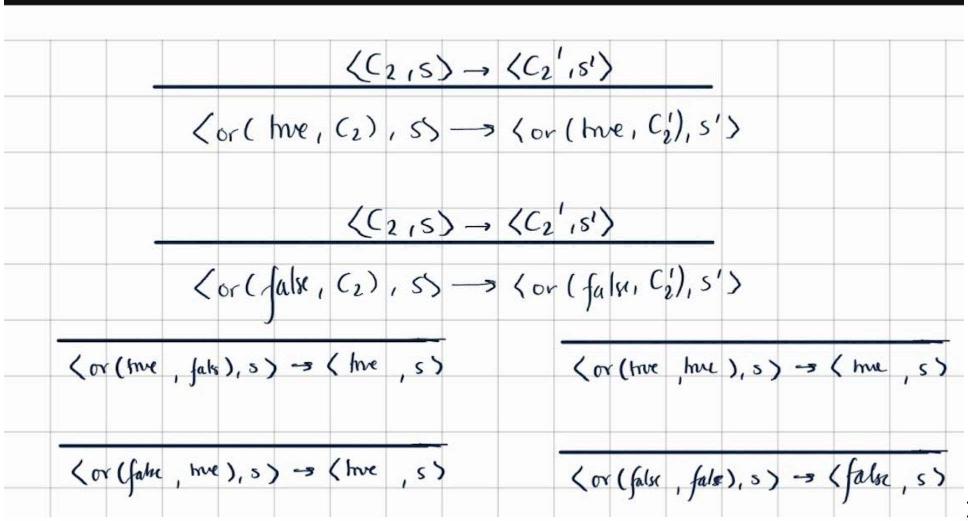
--------------------------------   --------------------

Or(C1, C2) -> Or(C1', C2)   Or(C1, C2) -> Or(C1, C2')



b) assume or is left associative.

$$\frac{\langle C_1, s\rangle \to \langle C_1', s'\rangle}{\langle or(C_1, C_2), s\rangle \to \langle or(C_1', C_2), s'\rangle}$$



$$\frac{\langle C_2, s\rangle \to \langle C_2', s'\rangle}{\langle or(true, C_2), s\rangle \to \langle or(true, C_2'), s'\rangle}$$

$$\frac{\langle C_2, s\rangle \to \langle C_2', s'\rangle}{\langle or(false, C_2), s\rangle \to \langle or(false, C_2'), s'\rangle}$$

$$\frac{}{\langle or(true, false), s\rangle \to \langle true, s\rangle} \qquad \frac{}{\langle or(true, true), s\rangle \to \langle true, s\rangle}$$

$$\frac{}{\langle or(false, true), s\rangle \to \langle true, s\rangle} \qquad \frac{}{\langle or(false, false), s\rangle \to \langle false, s\rangle}$$

11s

Why would a command eval to a boolean? Surely <or(C1, C2), s> -> <C1', s'> and the same for C2?

<loop(C), s> -> <C; loop(C), s>

C) Prove that the translation f preserves the meaning of commands: ∀i, C,s,s 0 . hC,si ⇓i s 0 ==⇒ hf(C),si ⇓ s 0 Do your proof using strong mathematical induction on i. You may also use the following lemma: ∀B,s : hB,si ⇓b false ==⇒ h¬B,si ⇓b true

HElP........ what is this

TODO: Someone upload a pic at some point. Also why did the question itself take up an entire page? I felt extremely attacked.

To note:

Question asks for strong mathematical induction on i.

The base case therefore has to include steps that take 0 derivations, and inductive steps will contain others.

Imo, base case: skip, x:=E, and while when B evaluates to false (because all of these contain (down arrow)0)

Inductive case: all others that include i

Q2

A) State your CID (yeah no), let’s call it 01234568. (also do one for odd CIDs?)

CID = 1234568 = 2^3 x 154321 = << 3, 77160 >>   --> the pair!

77160 = 2^3 x 9645 = << 3, 4822 >>

4822 = 2 x 2411 = << 1, 1205 >>

1205 = 2^0 x 1205 = << 0, 602 >>

602 = 2 x 301 = << 1, 150 >>

150 = 2 x 75 = << 1, 37 >>

37 = 2^0 x 37 = << 0, 18 >>

18 = 2 x 9 = << 1, 4 >> = << 1, << 2, 0>> >>

Combining everything:

1234568 = ‘[3, 1, 0, 1, 1, 0, 1, 2]’ ---> the list!

Now for the instruction. There is a typo (I think), so my interpretation was what is the I s.t. ‘I’ = CID.

(Rather than the entire program coded by the elements of the list above.)

Well, CID = << 3, 77160 >> , 3 is odd, so I is of the form ‘R_i^- -> L_j, L_k’

(where i = 1, <j,k> = 77160)

<j, k> = 2^j (2k + 1) - 1 = (77161) - 1 = 2^0 (2x 38580 + 1) - 1 = < 0, 38580 >

So, 1234568 = ‘R_1^- -> L_0, L_38580’ ---> the instruction!

F to people who did instructions for their lists, I certainly wrote 12+13 lines of instructions due to his lack of ability to make anything clear

REPEAT FOR CID + 8.

In principle, yes, (e)very student can decode their CIDs. This is because we were shown in lectures that << -, - >> : N x N -> N+ and < -, - > : N x N -> N and ' [-] ' : List N -> N are bijective functions.

So, '[body]' -> N and RM -> N are also bijections (compositions of bijections are also bijections)

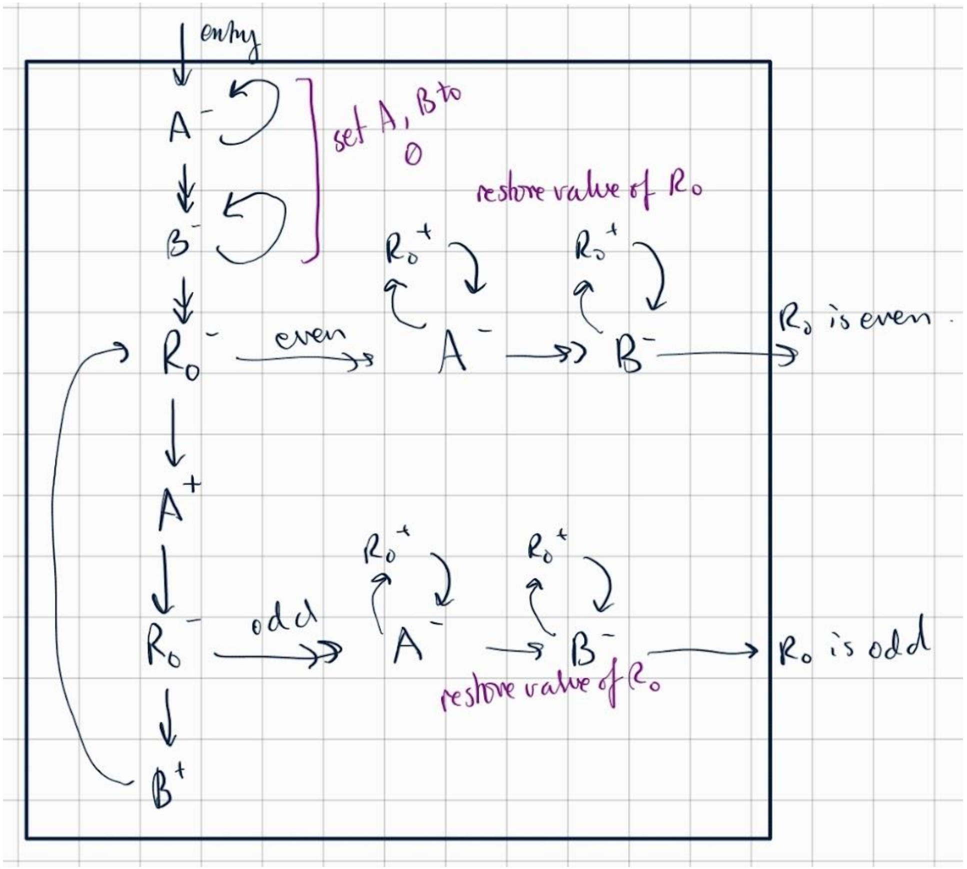
Thus, (by surjectivity) \forall CID.

\exists x, y. << x, y >> = CID AND \exists L. 'L' = CID (L used to denote loopy l)

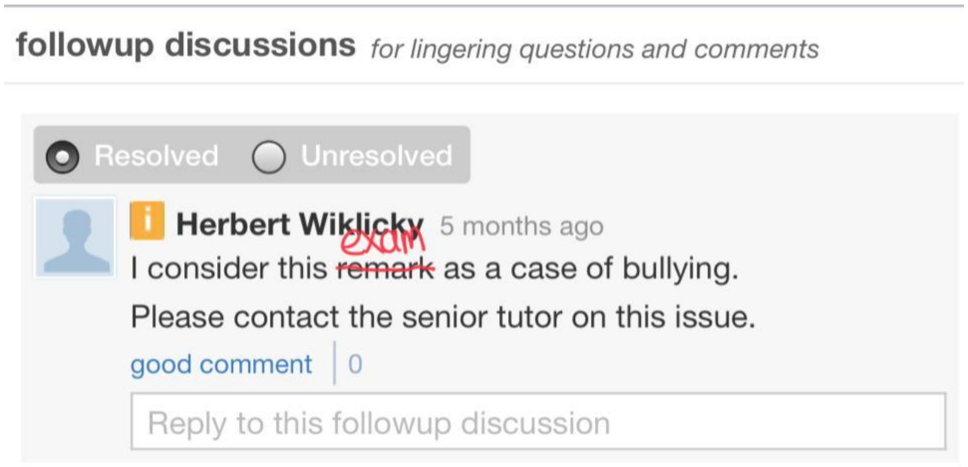
AND \exists Instr s.t. 'Instr' = CID

(same w.l.o.g for CID + 8)

2b)


entry
A⁻
B⁻
set A, B to 0
restore value of R₀
R₀⁺
R₀⁺
R₀⁻
even
A⁻
B⁻
R₀ is even.
A⁺
R₀⁺
R₀⁺
R₀⁻
odd
A⁻
B⁻
R₀ is odd
restore value of R₀
B⁺

Models Mark
There is a typo in the diagram in 2 b (ii)



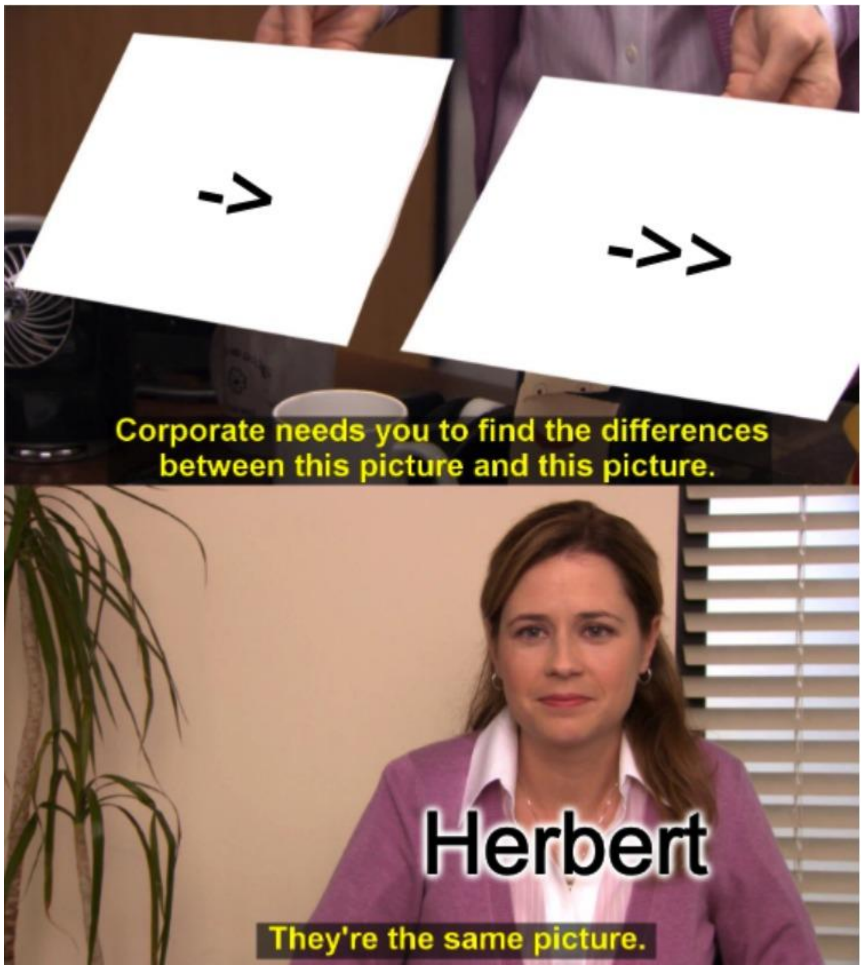
followup discussions *for lingering questions and comments*

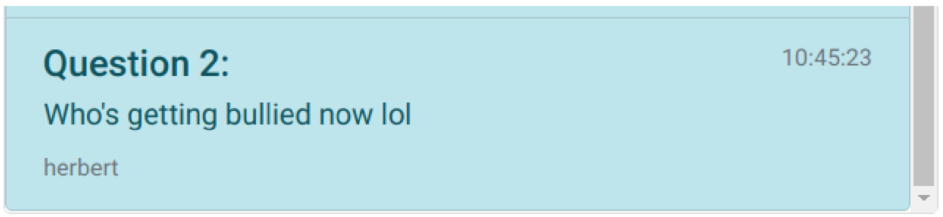
[✓ Resolved] Unresolved

Herbert Wiklicky 5 months ago
I consider this ~~remark~~ exam as a case of bullying.
Please contact the senior tutor on this issue.

good comment | 0

Reply to this followup discussion


->
->>
Corporate needs you to find the differences between this picture and this picture.
Herbert
They're the same picture.



Question 2: 10:45:23

Who's getting bullied now lol

herbert

2c)

Thanks be to Google and Wikipedia during this exam.

Were combinators even discussed in lectures? (Of course not) What are these?

$Y' = UU$

$U = \lambda ux\ x(uux)$

Well, U has no free variables. So, it is a combinator.

Additionally, Y' uses U only, which is a combinator, so no free variables are added.

Hence, Y' is also a combinator.

Let g be some random variable.

$Y'\ g = UU\ g$

$= (\lambda ux\ x(uux))\ U\ g$

$= (\lambda x\ x(UUx))\ g$

$= g\ (UUg) = g\ Y'\ g$

This is the property that says that Y' is a fixed-point combinator (?)