w1a) Do C first, and then keep executing command C until boolean B evaluates to true i.e. C; while not B, do C.

b)
(N.B. Petar mentioned there are two typos in the question paper in regard to this question. Firstly, it should be <B, s’> $\Downarrow_b$ True> etc. And the fact that we use $\Downarrow_b$ not $\Downarrow_c$ for boolean derivations)

<x := x + 2, s’’> $\Downarrow_c$ s’’’ <x > 3, s’’’> $\Downarrow_b$ True
----
<x := x + 2, s’> $\Downarrow_c$ s’’   <x > 3, s’’> $\Downarrow_b$ False   <C, s’’> $\Downarrow_c$ s’’’

<x := 0, s> $\Downarrow_c$ s’   <C, s’> $\Downarrow_c$ s’’’

<x := 0; C, s> $\Downarrow_c$ s’’’

C = repeat x := x + 2 until x > 3
s = [], s’ = [x -> 0], s’’ = [x -> 2], s’’’ = [x -> 4]

c)

Petar said you don’t need induction, just show a contradiction.

Proof using contradiction, similar to the one below:
Since Repeat and While are the same for expressions and booleans, ↓b and ↓e are deterministic. Need to prove then that ↓c is deterministic.

Assume ↓c is not deterministic towards a contradiction. Then forall C, s, s’,s’’ if <C, s> ↓ s’ and <C, s>↓ s’’ then s’ != s’’.

Assume <C, s> ↓ s’ and <C, s>↓ s’’.
By the lemma given, this gives us:
<f(C), s> ↓ s’ and <f(C), s>↓ s’’.
f(C) is a command in the While language, therefore by determinacy of While, we have s’ = s’’.
Contradiction, therefore Repeat is deterministic, as ↓b, ↓e and ↓c are.

Direct proof??

To show: ∀C, s, s’, s’’ . <C, s> ↓ s’ and <C, s> ↓ s’’ => s’ = s’’. Proof: Take arbitrary C, s, s’, s’’.

Then assume that <C, s> ↓ s’ and <C, s> ↓ s’’. Then, we have that <f(C), s> ↓ s’ and also <f(C), s> ↓ s’’. But since f(C) is a command in while, by determinacy of while we have that s’ = s’’, hence we are done.

Let R = repeat C until B.

Assume that <R, s> ↓ s^1 and <R,s> ↓ s^2.

By the given property of f, <f(R), s> ↓ s^1 and <f(R), s> ↓ s^2, which implies that $s^1 = s^2$ by the determinacy of while.

Therefore the repeat command is deterministic. Since all other repeat commands operate in the same way as they do in While, we now know that all Repeat commands are deterministic.

Base cases:
From the determinacy of While, we know **f(Skip)** and **f(x:=E)** are deterministic.

Inductive cases:
To show **f(C1;C2)** is deterministic: let **f(C1)** = **A** and **f(C2)** = **B**. From inductive hypothesis A and B are deterministic. By determinacy of while, **A;B** is deterministic. So **f(C1;C2)** is deterministic.

To show **f(if B then C1 else C2)** is deterministic: let **f(C1)** = **A** and **f(C2)** = **A2.** f(C1) and f(C2) are deterministic from the inductive hypothesis. By determinacy of while, **if B then A else A2** is deterministic.

If **B** evaluates to **b1** and **B** evaluates to **b2**, **b1 = b2** by determinacy of While.
So if **-B** evaluates to **b1** and **-B** evaluates to **b3**, **b4 = -b2 = -b1**
Let **f(C)** evaluate to **D**. By determinacy of While, **D** and so **f(C)** is deterministic.
while -B do C is also deterministic from the determinacy of While.
So **f(C); while -B do C** is also deterministic.

QED.

I don’t think you can just use the “magic formula by determinacy of while” every time. Like for the first two (base case) you can but then...

We can because f translates “from Repeat commands to While commands” and we know all While commands are deterministic. Also, for 10% of the paper, it’s very long.

Ok but then you still need to say that f(C1), f(C2)...are deterministic from base cases

Good point.

d)
i)

Is this rule included??

<skip, s> -> s

######

<E, s> -> <E’, s’>
<x:= E, s> -> <x:=E’, s’>

<x:= n, s> -> <skip, s[x|->n]

<c1,s> -> <c1’, s’>
<c1;c2,s> -> <c1’;c2,s’>

<skip;c1, s> -> <c1, s>

<B, s> -> <B’, s’>
<if B then c1 else c2, s> -> <if B’ then c1 else c2, s’>

<if true then c1 else c2, s> -> <c1, s>

<if false then c1 else c2, s> -> <c2, s>

<repeat C until B, s> -> <C; if B then skip else (repeat C until B), s>

ii)

Let P = repeat x = x + 2 until x > 3

<x = 0; P, s>
-> <skip; P, (x -> 0)>
-> <repeat x = x + 2 until x > 3, (x -> 0)>
-> <if x > 3 then skip else (x = x + 2; P), (x -> 0)>
-> <if false then skip else (x = x + 2; P), (x -> 0)>
-> <x = x + 2; P, (x -> 0)>
-> <x = 2; P, (x -> 0)>
-> <skip; P, (x -> 2)>
-> <repeat x = x + 2 until x > 3, (x -> 2)>
-> <if x > 3 then skip else (x = x + 2; P), (x -> 2)>
-> <if false then skip else (x = x + 2; P), (x -> 2)>
-> <x = x + 2; P, (x -> 2)>
-> <x = 4; P, (x -> 2)>
-> <skip; P, (x -> 4)>
-> <repeat x = x + 2 until x > 3, (x -> 4)>
-> <if x > 3 then skip else (x = x + 2; P), (x -> 4)>
-> <if true then skip else (x = x + 2; P), (x -> 4)>
-> <skip, (x -> 4)>
-> (x -> 4)

Alternative:
Let:
P = repeat x := x + 2 until x > 3
Q = if x>3 then skip else P
S_1 = S [x -> 0]
S_2 = S_1[x -> 2]
S_3 = S_2[x -> 4]

<x := 0; P, s> -> <skip; P, S_1>

-> <P, S_1>

-> <x := x + 2; Q, S_1>

-> <skip; Q, S_2>

-> <Q, S_2>

-> <if 2>3 then skip else P, S_2> // could skip to combine evaluation of E and B

-> <if false then skip else P, S_2>

-> <P, S_2>

-> <x := x + 2; Q, S_2>

-> <skip; Q, S_3>

-> <Q, S_3>

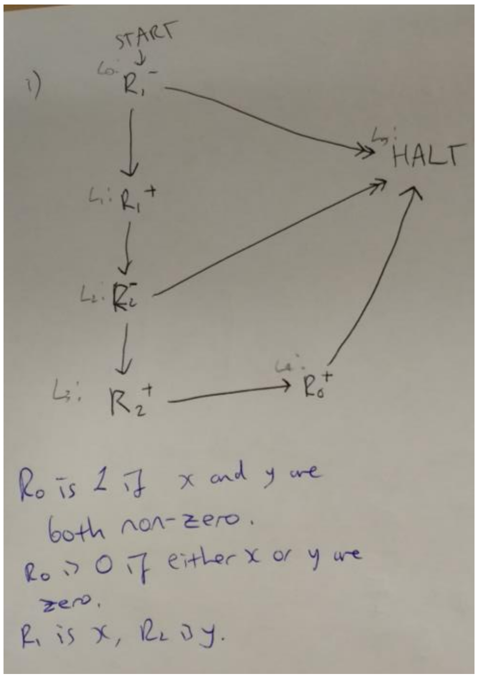
-> <if 4>3 then skip else P, S_3> // could skip to combine evaluation of E and B

-> <if true then skip else P, S_3>

-> <skip, S_3>

-> S_3

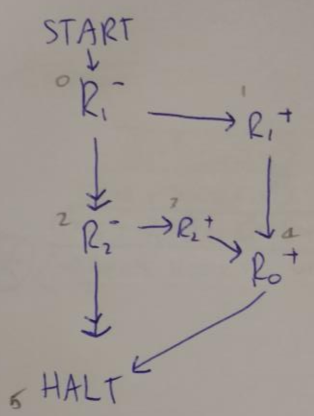
2a)
i)



R0 is 1 when both non zero (kinda an AND gate)

ii)

ii)



$L_0 : R_1^- \to L_1, L_2$

$L_1 : R_1^+ \to L_4$

$L_2 : R_2^- \to L_3, L_5$

$L_3 : R_2^+ \to \cancel{L} L_4$

$L_4 : R_0^+ \to L_5$

$L_5 :$ HALT

2b)

i)

IF TRUE $t f$

$=_\beta (\lambda b.\lambda t.\lambda f.\, b\, t\, f)(\lambda x.\lambda y.\, x)\, t\, f$

$=_\beta (\lambda x.\lambda y.\, x)\, t\, f$

$=_\beta t$

IF FALSE $t f$

$=_\beta (\lambda b.\lambda t.\lambda f.\, b\, t\, f)\, (\lambda x.\lambda y.\, \cancel{x}y)\, t\, f$

$=_\beta (\lambda x.\lambda y.\, y)\, t\, f$

$= f$

Mark scheme from Coursework 2:

Solution. (a) This part is straightforward β-reduction:

$$\begin{aligned}
\texttt{IF TRUE}\ t\ f &= (\lambda b.\lambda t.\lambda f.\ b\ t\ f)\ \texttt{TRUE}\ t\ f\\
&\to_\beta (\lambda t.\lambda f.\ \texttt{TRUE}\ t\ f)\ t\ f\\
&\to_\beta (\lambda f.\ \texttt{TRUE}\ t\ f)\ f\\
&\to_\beta \texttt{TRUE}\ t\ f\\
&= (\lambda x.\lambda y.\ x)\ t\ f\\
&\to_\beta (\lambda y.\ t)\ f\\
&\to_\beta t
\end{aligned}$$

or, alternatively

$$\begin{aligned}
\texttt{IF TRUE}\ t\ f &= (\lambda b.\lambda t.\lambda f.\ b\ t\ f)\ \texttt{TRUE}\ t\ f\\
&\to_\beta (\lambda t.\lambda f.\ \texttt{TRUE}\ t\ f)\ t\ f\\
&= (\lambda t.\lambda f.\ (\lambda x.\lambda y.\ x)\ t\ f)\ t\ f\\
&\to_\beta (\lambda t.\lambda f.\ (\lambda y.\ t)\ f)\ t\ f\\
&\to_\beta (\lambda t.\lambda f.\ t)\ t\ f\\
&\to_\beta (\lambda f.\ t)\ f\\
&\to_\beta t
\end{aligned}$$

Similarly,

$$\begin{aligned}
\texttt{IF FALSE}\ t\ f &= (\lambda b.\lambda t.\lambda f.\ b\ t\ f)\ \texttt{FALSE}\ t\ f\\
&\to_\beta (\lambda t.\lambda f.\ \texttt{FALSE}\ t\ f)\ t\ f\\
&\to_\beta (\lambda f.\ \texttt{FALSE}\ t\ f)\ f\\
&\to_\beta \texttt{FALSE}\ t\ f\\
&= (\lambda x.\lambda y.\ y)\ t\ f\\
&\to_\beta (\lambda y.\ y)\ f\\
&\to_\beta f
\end{aligned}$$

ii)

or, alternatively

$$\begin{array}{rcl} \texttt{IF FALSE}\ t\ f & = & (\lambda b.\lambda t.\lambda f.\ b\ t\ f)\ \texttt{FALSE}\ t\ f \\ & \to_\beta & (\lambda t.\lambda f.\ \texttt{FALSE}\ t\ f)\ t\ f \\ & = & (\lambda t.\lambda f.\ (\lambda x.\lambda y.\ y)\ t\ f)\ t\ f \\ & \to_\beta & (\lambda t.\lambda f.\ (\lambda y.\ y)\ f)\ t\ f \\ & \to_\beta & (\lambda t.\lambda f.\ f)\ t\ f \\ & \to_\beta & (\lambda f.\ f)\ f \\ & \to_\beta & f \end{array}$$

[4 marks]

(b) This part is also straightforward, given that we know how IF behaves from part (a).

$$\begin{array}{rcl} \texttt{NOT (NOT TRUE)} & = & \texttt{NOT}\ ((\lambda b.\ \texttt{IF}\ b\ \texttt{FALSE TRUE})\ \texttt{TRUE}) \\ & \to_\beta & \texttt{NOT (IF TRUE FALSE TRUE)} \\ & \to_\beta & \texttt{NOT FALSE} \\ & = & (\lambda b.\ \texttt{IF}\ b\ \texttt{FALSE TRUE})\ \texttt{FALSE} \\ & \to_\beta & \texttt{IF FALSE FALSE TRUE} \\ & \to_\beta & \texttt{TRUE} \end{array}$$

$$\begin{array}{rcl} \texttt{IF FALSE}\ t\ f & = & (\lambda b.\lambda t.\lambda f.\ b\ t\ f)\ \texttt{FALSE}\ t\ f \\ & \to_\beta & (\lambda t.\lambda f.\ \texttt{FALSE}\ t\ f)\ t\ f \\ & \to_\beta & (\lambda f.\ \texttt{FALSE}\ t\ f)\ f \\ & \to_\beta & \texttt{FALSE}\ t\ f \\ & = & (\lambda x.\lambda y.\ y)\ t\ f \\ & \to_\beta & (\lambda y.\ y)\ f \\ & \to_\beta & f \end{array}$$

iii)

AND ≜ λa.λb.(IF a b FALSE)

OR ≜ λa.λb. IF a TRUE b

(without redundant IF)

AND ≜ λa.λb.(a b FALSE)

OR ≜ λa.λb.(a TRUE b)

Or better:

AND ≜ λa.λb.(a b a)

OR ≜ λa.λb.(a a b)