(Brock-Ackerman) The following picture shows a network of communicating processes.

![Diagram of a network of communicating processes]

The formal description of this network is

\[
\text{chan } a, b, c \mid \text{choose } \mid (c?. b! c)
\]

Formally define \text{choose} as follows:

\[
\text{choose } \equiv (a?. (c! 0 \parallel d! 0)) \lor (b?. (c! 1 \parallel d! 1))
\]

Now the network is

\[
\text{chan } a, b, c \mid \text{choose } \mid (c?. b! c)
\]

\[
= \exists \text{hd}, \text{ra}, \text{ra}', \text{wa}, \text{wa}', \text{hb}, \text{rb}, \text{rb}', \text{wb}, \text{wb}', \text{hc}, \text{rc}, \text{rc}', \text{wc}, \text{we}'.
\]

\[
\exists \text{ta}, \text{tc}, \text{tb}.
\]

\[
\begin{align*}
\text{ta} & = \text{ra}_0 = t \land \text{hd}_0 = 0 \land \text{wa}' = 1 \\
\land ( & \text{tc} = \text{rc}_0 = \text{td}_w = \text{ra}_0 + 1 \land \text{ra}' = \text{wc}' = 1 \land \text{hd}_0 = \text{hd}_w = 0 \land \text{wd}' = \text{wd} + 1 \\
\lor & \text{tc} = \text{rc}_0 = \text{td}_w = \text{rb}_0 + 1 \land \text{rb}' = \text{wc}' = 1 \land \text{hd}_0 = \text{hd}_w = 0 \land \text{wd}' = \text{wd} + 1) \\
\land & \text{tb} = \text{rb}_0 = \text{rc}_0 + 1 \land \text{rc}' = \text{wb}' = 1 \land \text{hd}_0 = \text{hd}_c = 0 \\
\land & \text{t}' = \text{MAX} [\{\text{ta}; \text{tc}; \text{tb}\}] \quad \text{use One-Point laws to eliminate most quantifiers.}
\end{align*}
\]

\[
\exists \text{tc}, \text{tb}.
\]

\[
\begin{align*}
\text{tc} & = \text{td}_w = t + 1 \land \text{hd}_w = 0 \land \text{wd}' = \text{wd} + 1 \\
\lor & \text{tc} = \text{td}_w = \text{tb} + 1 \land \text{hd}_w = 1 \land \text{wd}' = \text{wd} + 1) \\
\land & \text{tb} = \text{tc} + 1 \\
\land & \text{t}' = \text{MAX} [\{\text{tc}; \text{tb}\}] \quad \text{move the conjunctions into the disjunction}
\end{align*}
\]

\[
\exists \text{tc}, \text{tb}.
\]

\[
\begin{align*}
\text{tc} & = \text{td}_w = t + 1 \land \text{hd}_w = 0 \land \text{wd}' = \text{wd} + 1 \land \text{tb} = \text{tc} + 1 \land \text{t}' = \text{MAX} [\{\text{tc}; \text{tb}\}] \\
\lor & \text{tc} = \text{td}_w = \text{tb} + 1 \land \text{hd}_w = 0 \land \text{wd}' = \text{wd} + 1 \land \text{tb} = \text{tc} + 1 \land \text{t}' = \text{MAX} [\{\text{tc}; \text{tb}\}] \\
\text{now we can eliminate } & \text{tc} \text{ and } \text{tb} \text{ in each disjunct separately}
\end{align*}
\]

\[
\begin{align*}
\text{td}_w & = t + 1 \land \text{hd}_w = 0 \land \text{wd}' = \text{wd} + 1 \land \text{t}' = t + 2 \\
\lor & \text{td}_w = \infty \land \text{hd}_w = 1 \land \text{wd}' = \text{wd} + 1 \land \text{t}' = \infty
\end{align*}
\]

\[
(t := t + 1. \quad d! 0. \quad t := t + 1) \lor (t := \infty. \quad d! 1)
\]

There is probably a better way to do this question by using laws of programs and not translating to ordinary logic.
(b) As in part (a), \textit{choose} either reads from \( a \) and then outputs a 0 on \( c \) and \( d \), or reads from \( b \) and then outputs a 1 on \( c \) and \( d \). But this time the choice is not made freely; \textit{choose} reads from the channel whose input is available first (if there’s a tie, then take either one).

§

We define \textit{choose} as follows:

\[
\text{choose } = (\sqrt{a} \lor \nu a \leq \nu b_r) \land (\nu c \cdot (c! \parallel d!)) \lor (\sqrt{b} \lor \nu b_r \leq \nu a_r) \land (b? \cdot (c! \parallel d!))
\]

Now the network is

\[
\text{chan } a, b, c \quad a! 0 \parallel \text{choose } \parallel (c? \cdot b! c)
\]

\[
= \exists b_a, \nu a, ra, ra', wa, wa', wb, T_b, rb, rb', wb, wb', \nu M_c, \nu T_c, rc, rc', wc, wc'.
\]

\[
ra := 0. \quad wa := 0. \quad rb := 0. \quad wb := 0. \quad rc := 0. \quad wc := 0.
\]
 \[
( b_a = 0 \land \nu a = t \land (wa := wa+1) \\
\lor ( ( \sqrt{t_a} \leq t \lor \nu t_a \leq \nu T_{rb}) \\
\land (t := \max t \nu t_a + 1). \quad ra := ra+1. \\
( \nu M_c = 0 \land \nu wc = t \land (wc := wc+1) \\
\lor ( \nu M_d = 0 \land \nu T_d = t \land (wd := wd+1). ))
\lor ( ( \sqrt{t_b} \leq t \lor \nu T_{rb} \leq \nu t_r) \\
\land (t := \max t \nu t_b + 1). \quad rb := rb+1. \\
( \nu M_c = 1 \land \nu wc = t \land (wc := wc+1) \\
\lor ( \nu M_d = 1 \land \nu T_d = t \land (wd := wd+1)). )))
\]

\[
= \exists b_a, \nu a, ra, ra', wa, wa', wb, T_b, rb, rb', wb, wb', \nu M_c, \nu T_c, rc, rc', wc, wc'.
\]

\[
\exists t_a, t_c, t_b
\]

\[
t_a = t_0 = t \land b_a = 0 \land wa' = 1
\land ( ( \sqrt{t_0} \leq t \lor \nu t_0 \leq \nu b_0) \\
\land t_c = t_0 = t_d = t_0 + 1 \land ra' = wc' = 1 \land \nu M_c = M_d = 0 \land wd' = wd + 1
\lor ( \sqrt{t_0} \leq t \lor \nu t_0 \leq \nu t_0) \\
\land t_c = t_0 = t_d = t_0+1 \land rb' = wc' = 1 \land \nu M_c = M_d = 1 \land wd' = wd + 1)
\land t_b = t_0 = t_0 + 1 \land rc' = wb' = 1 \land \nu M_b = M_c
\land t' = \text{MAX} \, [t_a; t_c; t_b]
\]

use the One-Point laws to eliminate most quantifiers

\[
= \exists t_c, t_b
\]

\[
( ( t \leq t' \lor t \leq t_b) \land t_c = t_d = t+1 \land \nu M_d = 0 \land wd' = wd + 1 \\
\lor ( t \leq t \lor t_b \leq t) \land t_c = t_d = t_b + 1 \land \nu M_d = 1 \land wd' = wd + 1)
\land t_b = t_c + 1
\land t' = \text{MAX} \, [t; t_c; t_b]
\]

simplify the two minor disjunctions
and move the conjunctions into the major disjunction

\[
= \exists t_c, t_b
\]

\[
t_c = t_d = t+1 \land \nu M_d = 0 \land wd' = wd + 1 \land t_b = t_c + 1 \land t' = \text{MAX} \, [t; t_c; t_b]
\lor (t_b \leq t \land t_c = t_d = t_b + 1 \land \nu M_d = 1 \land wd' = wd + 1 \land t_b = t_c + 1 \land t' = \text{MAX} \, [t; t_c; t_b]
\]

now we can eliminate \( t_c \) \quad and \( t_b \) in each disjunct separately

\[
T_d = t+1 \land \nu M_d = 0 \land wd' = wd + 1 \land t' = t + 2
\lor \infty \leq t \land T_d = \infty \land \nu M_d = 1 \land wd' = wd + 1 \land t' = \infty.
\]

\[
= (t := t+1. \quad d! 0. \quad t := \infty \lor (d!))
\]

If the computation starts before time \( \infty \), the output is definitely 0. Again, there is probably a better way to do this question by using laws of programs and not translating to ordinary logic.