(choose) The following picture shows a network of communicating processes.

```
| d! 0 |
| a |
| c |
| choose |
```

The formal description of this network is

```
chan a, b, c. a! 0 || choose || (c?. b! c)
```

Formally define `choose`, add transit time, and state the output message and time if

(a) `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`. The choice is made freely.

(b) as in part (a), `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; `choose` reads from the channel whose input is available first (if there's a tie, then take either one).

After trying the question, scroll down to the solution.
(a) 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 $d$. The choice is made freely.

§ We define choose as follows:

$$\text{choose} \equiv (a? \cdot (c! 0 \parallel d! 0)) \lor (b? \cdot (c! 1 \parallel d! 1))$$

Now we calculate.

$$\text{chan } a, b, c \cdot a! 0 \parallel \text{choose } \parallel (c?. b! c)$$

$$= \exists \mathcal{M}, \mathcal{T}, n, r_a, r'_a, w_a, w'_a, \mathcal{M}_b, \mathcal{T}_b, r_b, r'_b, w_b, w'_b, \mathcal{M}_c, \mathcal{T}_c, r_c, r'_c, w_c, w'_c.

(\forall i, j \cdot i \neq j \Rightarrow t \leq \mathcal{T}_i \leq \mathcal{T}_j \leq t' \land t \leq \mathcal{T}_i \leq \mathcal{T}_j \leq t' \land t \leq \mathcal{T}_i \leq \mathcal{T}_j \leq t')$$

$$\land r_a = w_a = r_b = w_b = n = w_c = 0$$

$$\land (\mathcal{M}_a = 0 \land \mathcal{T}_a = t \land (w_a = w_a + 1))$$

$$\lor (t = t_1 \cdot (\mathcal{T}_r + 1). r_v = r_v + 1.$$

$$\land (\mathcal{M}_w = 0 \land \mathcal{T}_w = t \land (w_d = w_d + 1))$$

$$\lor (t = t_1 \cdot (\mathcal{T}_r + 1). r_c = r_c + 1.$$}

$$\land (\mathcal{M}_b = 0 \land \mathcal{T}_b = t \land (w_b = w_b + 1))$$

Except for time, all processes in concurrent compositions change different variables, so || is easily replaced by conjunction.

Also, make all substitutions indicated by assignments.

$$= \exists \mathcal{M}, \mathcal{T}, n, r_a, r'_a, w_a, w'_a, \mathcal{M}_b, \mathcal{T}_b, r_b, r'_b, w_b, w'_b, \mathcal{M}_c, \mathcal{T}_c, r_c, r'_c, w_c, w'_c.$$

(\forall i, j \cdot i \neq j \Rightarrow t \leq \mathcal{T}_i \leq \mathcal{T}_j \leq t' \land t \leq \mathcal{T}_i \leq \mathcal{T}_j \leq t' \land t \leq \mathcal{T}_i \leq \mathcal{T}_j \leq t')$$

$$\land r_a = w_a = r_b = w_b = n = w_c = 0$$

$$\land \exists \mathcal{T}, t_c, t_b.$$

$$t_a = \mathcal{T}_0 = t \land \mathcal{M}_0 = 0 \land w_d = 1$$

$$\land \left( t = \mathcal{T}_0 = \mathcal{T}_d = \mathcal{T}_0 + 1 \land r_a = w_c' = 1 \land \mathcal{M}_0 = \mathcal{M}_d = 0 \land w_d = w_d + 1 \right.$$

$$\lor t = \mathcal{T}_0 = \mathcal{T}_d = \mathcal{T}_0 + 1 \land r_b = w_c' = 1 \land \mathcal{M}_0 = \mathcal{M}_d = 1 \land w_d = w_d + 1 $$

$$\land t_b = \mathcal{T}_0 + 1 \land r_c' = w_b = 1 \land \mathcal{M}_0 = \mathcal{M}_d$$

$$\land t' = t \cdot t_1 \cdot t_b$$

use One-Point laws to eliminate most quantifiers

$$= \exists \mathcal{T}, t_c, t_b.$$

$$t_c = \mathcal{T}_d = t + 1 \land \mathcal{M}_d = 0 \land w_d = w_d + 1$$

$$\lor t_c = \mathcal{T}_d = t + 1 \land \mathcal{M}_d = 1 \land w_d = w_d + 1 $$

$$\land t_b = t_c + 1$$

$$\land t' = t \cdot t_1 \cdot t_b$$

move the conjunctions into the disjunction

$$= \exists \mathcal{T}, t_c, t_b.$$

$$t_c = \mathcal{T}_d = t + 1 \land \mathcal{M}_d = 0 \land w_d = w_d + 1 \land t_b = t_c + 1 \land t' = t \cdot t_1 \cdot t_b$$

now we can eliminate $t_c$ and $t_b$ in each disjunct separately

$$= \mathcal{T}_d = t + 1 \land \mathcal{M}_d = 0 \land w_d = w_d + 1 \land t' = t + 2$$

$$\lor \mathcal{T}_d = t + 1 \land \mathcal{M}_d = 1 \land w_d = w_d + 1 \land t = \infty$$

$$(t := t + 1, d \cdot 0, t := t + 1) \lor (t := \infty, d \cdot 1)$$

Either a 0 is output after time 1 or nothing ever happens. 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), 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; choose reads from the channel whose input is available first (if there's a tie, then take either one).

§ There is a slight ambiguity in the question. It says “the channel whose input is available first”. Does this mean the channel whose input arrived first? Or, if two inputs have
already arrived, no matter which arrived first, they are both available now (at the same
time)? I'll do it both ways. Formalizing makes the meaning clear.

Suppose “available first” means “arrived first”. We define choose as follows:

\[
\text{choose } = \begin{cases} 
T_{t_a} \leq T_{t_b} \land (a? \cdot (c! 0 \parallel d! 0)) \\
\lor T_{t_b} \leq T_{t_a} \land (b? \cdot (c! 1 \parallel d! 1))
\end{cases}
\]

Now we calculate.

\[
\begin{align*}
\text{chan a, b, c } a! 0 \parallel \text{ choose } \parallel (c?. b! c) \\
= \text{ exist } Ma, Fa, ra, ra', wa, wa', Mb, Fb, rb, rb', wb, wb', Mc, Fc, rc, rc', wc, wc'. \\
(\forall i, j; i \leq j \Rightarrow t \leq T_i \leq T_j \leq t' \land t \leq T_i \leq T_j \leq t' \land T_i \leq T_j \leq t') \\
\land ra = wa = rb = wb = rc = wc = 0 \\
\land (Mb_{we} = 0 \land Ta_{we} = 1 \land (wa = wc + 1)) \\
\land \begin{cases} 
T_{a_{t_b}} \leq T_{a_{t_b}} \\
(t := t' \uparrow (Fa_{t_b} + 1)). ra := ra + 1.
\end{cases} \\
\land (Mb_{we} = 1 \land Ta_{we} = t \land (wa = wc + 1)) \\
\land \begin{cases} 
Mb_{we} = Mb_{we} - 1 \land Tb_{we} = t \land (wb = wb + 1)) \\
(t := t' \uparrow (Fc_{rc} + 1)). r_c := r_c + 1.
\end{cases}
\end{align*}
\]

Except for time, all processes in concurrent compositions change different variables, so \( \parallel \) is easily replaced by conjunction.

Also, make all substitutions indicated by assignments.

\[
\begin{align*}
= \text{ exist } Ma, Fa, ra, ra', wa, wa', Mb, Fb, rb, rb', wb, wb', Mc, Fc, rc, rc', wc, wc'. \\
(\forall i, j; i \leq j \Rightarrow t \leq T_i \leq T_j \leq t' \land t \leq T_i \leq T_j \leq t' \land T_i \leq T_j \leq t') \\
\land ra = wa = rb = wb = rc = wc = 0 \\
\land \begin{cases} 
ta = Ta_{t_b} = 0 \land wa' = 1 \\
(Ta_{t_b} \leq T_{a_{t_b}}) \\
tc = Tc_{t_c} = Mb_{we} = tc + 1 \land rc' = wc' = 1 \land Mb_{we} = Mb_{we} = 0 \land wd' = wd + 1 \\
(Tb_{we} \leq Tc_{t_c}) \\
tb = Tb_{we} = Tb_{we} = wc' = 1 \land Mb_{we} = Mb_{we} = 1 \land tc = tc = tc + 1 \land rc' = wc' = 1 \\
(t' = tc \uparrow tc \uparrow tb)
\end{cases}
\end{align*}
\]

use the One-Point laws to eliminate most quantifiers

\[
\begin{align*}
= \text{ exist } tc, tb. \\
( \begin{cases} 
tb \leq tc \land tc = Tc_{t_b} = t + 1 \land Mb_{we} = 0 \land wd' = wd + 1 \\
\lor \begin{cases} 
tb \leq tc \land tc = Tc_{t_b} = tb + 1 \land Mb_{we} = 1 \land wd' = wd + 1 \\
\end{cases}
\end{cases} \\
\land \begin{cases} 
tb = tc + 1 \\
tc = tc \uparrow tc \uparrow tb \\
t' = t \uparrow tc \uparrow tb
\end{cases}
\end{align*}
\]

move the conjunctions into the disjunction

\[
\begin{align*}
= \text{ exist } tc, tb. \\
\begin{cases} 
tc = Tc_{t_b} = t + 1 \land Mb_{we} = 0 \land wd' = wd + 1 \land tb = tc + 1 \land t' = t \uparrow tc \uparrow tb \\
\lor \begin{cases} 
tb = tc \uparrow tc \uparrow tb \\
tc = Tc_{t_b} = tb + 1 \land Mb_{we} = 1 \land wd' = wd + 1 \land tb = tc + 1 \land t' = t \uparrow tc \uparrow tb
\end{cases}
\end{cases}
\end{align*}
\]

we can eliminate \( tc \) and \( tb \) in each disjunct separately

\[
\begin{align*}
= Tc_{t_b} = t + 1 \land Mb_{we} = 0 \land wd' = wd + 1 \land t' = t + 2 \\
\lor \begin{cases} 
\infty \leq t \land Tc_{t_b} = \infty \land Mb_{we} = 1 \land wd' = wd + 1 \land t' = \infty
\end{cases}
\end{align*}
\]

If the computation starts before time \( \infty \) the output is definitely \( 0 \) after time \( 1 \). Again, there is probably a better way to do this question by using laws of programs and not translating to ordinary logic.
Suppose “available first” means “if it’s available now, it doesn’t matter when it arrived”.

We define choose as follows:

\[
\text{choose } = \begin{cases} \\
\sqrt{a} \lor (T_{\text{if}} \leq T_{\text{rb}}) \land (a \vdash (c! \parallel d!)) \lor (b \vdash (T_{\text{rb}} \leq T_{\text{if}}) \land (b \vdash (c! \parallel d!)))
\end{cases}
\]

Now we calculate.

\[
\text{chan } a, b, c \vdash a! 0 \parallel \text{choose } \parallel (c!. b! c)
\]

\[
= \exists Ma, Ta, ra, ra’, wa, wa’, Mb, Tb, rb, rb’, wb, wb’, Mc, Pc, rc, rc’, wc, wc’.
\]

\[
(\forall i, j \leq i \Rightarrow t \leq T_{ai} \leq T_{aj} \leq t’ \land t \leq T_{bi} \leq T_{bj} \leq t’ \land t \leq T_{ci} \leq T_{cj} \leq t’)
\]

\[
\land ra = wa = rb = wb = wc = 0
\]

\[
\land (M_{\text{if}} = 0 \land T_{\text{if}} = t \land (wc := wc + 1))
\]

\[
\lor (T_{\text{if}} = t \lor T_{\text{if}} \leq T_{\text{rb}})
\]

\[
\land (r := ra + 1).
\]

\[
(\exists r := ra + 1.
\]

\[
= \exists Ma, Ta, ra, ra’, wa, wa’, Mb, Tb, rb, rb’, wb, wb’, Mc, Pc, rc, rc’, wc, wc’.
\]

\[
(\forall i, j \leq i \Rightarrow t \leq T_{ai} \leq T_{aj} \leq t’ \land t \leq T_{bi} \leq T_{bj} \leq t’ \land t \leq T_{ci} \leq T_{cj} \leq t’)
\]

\[
\land ra = wa = rb = wb = wc = 0
\]

\[
\land \exists ta, tc, tb.
\]

\[
= \exists ta, tc, tb.
\]

\[
= \exists tc, tb.
\]

\[
= \exists tc, tb.
\]

\[
\]

Except for time, all processes in concurrent compositions change different variables, so || is easily replaced by conjunction.

Also, make all substitutions indicated by assignments.

\[
= \exists tc, tb.
\]

\[
= \exists tc, tb.
\]

\[
= \exists tc, tb.
\]

If the computation starts before time \( \infty \) the output is definitely 0 after time 1. This is the same as before, so the ambiguity didn’t matter. This is good, because our programming constructs do not require us to keep track of the time messages arrive.
Again, there is probably a better way to do this question by using laws of programs and not translating to ordinary logic.