Distributed Processes, Types, and Programming

Sergey Ichtchenko

Mini-project, Michaelmas Term 2024

Question 1

(a) Give the free names and variables of the following processes.

(1) $(\nu\, d)(x(y).\overline{z}\langle d\rangle) \mid !\, e(y).\overline{y}\langle y\rangle \mid !\, d(y).(\nu\, c)\overline{y}\langle c\rangle$

(2) $c(z).(\nu\, e)(\nu\, a)\overline{e}\langle z\rangle \mid ((\nu\, a, b)y(x).\overline{b}\langle x\rangle \mid \overline{b}\langle a\rangle \mid \overline{e}\langle x\rangle)$

(1) The free names are d and e. The free variables are x and z.

Explanation:

$$
\begin{aligned}
&f_n((\nu\, d)(x(y).\overline{z}\langle d\rangle) \mid !\, e(y).\overline{y}\langle y\rangle \mid !\, d(y).(\nu\, c)\overline{y}\langle c\rangle) = \\
&= f_n((\nu\, d)(x(y).\overline{z}\langle d\rangle)) \cup f_n(!\, e(y).\overline{y}\langle y\rangle) \cup f_n(!\, d(y).(\nu\, c)\overline{y}\langle c\rangle)) = \\
&= (f_n(x(y).\overline{z}\langle d\rangle) \setminus \{d\}) \cup f_n(e) \cup f_n(\overline{y}\langle y\rangle) \cup f_n(d) \cup (f_n(\overline{y}\langle c\rangle) \setminus \{c\}) = \\
&= \varnothing \cup \{e\} \cup \varnothing \cup \{d\} \cup \varnothing = \{d, e\}
\end{aligned}
$$

$$
\begin{aligned}
&f_v((\nu\, d)(x(y).\overline{z}\langle d\rangle) \mid !\, e(y).\overline{y}\langle y\rangle \mid !\, d(y).(\nu\, c)\overline{y}\langle c\rangle) = \\
&= f_v((\nu\, d)(x(y).\overline{z}\langle d\rangle)) \cup f_v(!\, e(y).\overline{y}\langle y\rangle) \cup f_v(!\, d(y).(\nu\, c)\overline{y}\langle c\rangle) = \\
&= f_v(x(y).\overline{z}\langle d\rangle) \cup f_v(e(y).\overline{y}\langle y\rangle) \cup f_v(d(y).(\nu\, c)\overline{y}\langle c\rangle) = \\
&= (f_v(x) \cup f_v(\overline{z}\langle d\rangle) \setminus \{y\}) \cup (f_v(e) \cup f_v(\overline{y}\langle y\rangle) \setminus \{y\}) \cup (f_v(d) \cup f_v((\nu\, c)\overline{y}\langle c\rangle) \setminus \{y\}) = \\
&= (\{x\} \cup \{z\} \setminus \{y\}) \cup (\varnothing \cup \{y\} \setminus \{y\}) \cup (\varnothing \cup \{y\} \setminus \{y\}) = \{x, z\}
\end{aligned}
$$

□

(2) The free names are a, b, c, and e. The free variables are x and y.

Explanation:

$$
\begin{aligned}
&f_n(c(z).(\nu\, e)(\nu\, a)\overline{e}\langle z\rangle \mid ((\nu\, a, b)y(x).\overline{b}\langle x\rangle \mid \overline{b}\langle a\rangle \mid \overline{e}\langle x\rangle)) = \\
&= f_n(c(z).(\nu\, e)(\nu\, a)\overline{e}\langle z\rangle) \cup f_n((\nu\, a, b)y(x).\overline{b}\langle x\rangle \mid \overline{b}\langle a\rangle \mid \overline{e}\langle x\rangle) = \\
&= f_n(c) \cup (f_n(\overline{e}\langle z\rangle) \setminus \{a, e\}) \cup f_n((\nu\, a, b)y(x).\overline{b}\langle x\rangle) \cup f_n(\overline{b}\langle a\rangle) \cup f_n(\overline{e}\langle x\rangle) = \\
&= \{c\} \cup (\{e\} \setminus \{a, e\}) \cup (f_n(y(x).\overline{b}\langle x\rangle) \setminus \{a, b\}) \cup \{b, a\} \cup \{e\} = \\
&= \{c\} \cup (\{e\} \setminus \{a, e\}) \cup (\{b\}) \setminus \{a, b\}) \cup \{b, a\} \cup \{e\} = \\
&= \{c\} \cup \varnothing \cup \varnothing \cup \{b, a\} \cup \{e\} = \{a, b, c, e\}
\end{aligned}
$$

$$
\begin{aligned}
& f_v(c(z).(\nu\, e)(\nu\, a)\overline{e}\langle z\rangle \mid ((\nu\, a, b)y(x).\overline{b}\langle x\rangle \mid \overline{b}\langle a\rangle \mid \overline{e}\langle x\rangle)) = \\
&= f_v(c(z).(\nu\, e)(\nu\, a)\overline{e}\langle z\rangle) \cup f_v((\nu\, a, b)y(x).\overline{b}\langle x\rangle \mid \overline{b}\langle a\rangle \mid \overline{e}\langle x\rangle) = \\
&= (f_v(c) \cup f_v((\nu\, e)(\nu\, a)\overline{e}\langle z\rangle) \setminus \{z\}) \cup f_v((\nu\, a, b)y(x).\overline{b}\langle x\rangle) \cup f_v(\overline{b}\langle a\rangle) \cup f_v(\overline{e}\langle x\rangle) = \\
&= (\varnothing \cup f_v(\overline{e}\langle z\rangle) \setminus \{z\}) \cup f_v(y(x).\overline{b}\langle x\rangle) \cup \varnothing \cup \{x\} = \\
&= (\{z\} \setminus \{z\}) \cup (f_v(y) \cup f_v(\overline{b}\langle x\rangle) \setminus \{x\}) \cup \{x\} = \\
&= \varnothing \cup (\{y\} \cup \{x\} \setminus \{x\}) \cup \{x\} = (\{y\} \cup \varnothing) \cup \{x\} = \{x, y\}
\end{aligned}
$$

□

(b) This question is about the monadic asynchronous π-calculus.

(1) Define the size of the context C (denoted by size(C)) size : $\mathcal{P} \cup \{-\} \to \mathbb{N}$ assuming size($-$) = 1.

(2) Assume $P = a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle)$ and $P' = a(x).(\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle)$.
Are P and P' reduction congruent?

(3) Assume $Q =! a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle)$ and $Q' =! a(x).(\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle)$.
Are Q and Q' reduction congruent?

(1) Let us define size(C) for a context C in a similar way to size(P) defined in the question:

$$\begin{cases} \text{size}(-) = 1 \\ \text{size}(C \mid P) = \text{size}(C) + \text{size}(P) \\ \text{size}(P \mid C) = \text{size}(P) + \text{size}(C) \\ \text{size}((\nu\, a)C) = 1 + \text{size}(C) \end{cases}$$

(2) P and P' are **not** reduction congruent. Proof:
The context $C = -|\overline{a}\langle x\rangle|a(x).a(x).\overline{b}\langle x\rangle$ separates P and P'. Observe that P can reduce to the following expressions. *Note: the numbers under the arrows indicate separate reduction paths. For example, after the first line, there are two possible ways in which the reduction can go, indicated by arrow 1 and arrow 2. While arrow 1 terminates, there are again two possibilities after arrow 2, denoted with arrows 2.1 and 2.2.*

$$\begin{aligned} C[P] &= a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\xrightarrow[1]{} a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid a(x).\overline{b}\langle x\rangle \\ &\xrightarrow[2]{} a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\quad\xrightarrow[2.1]{} a(x).\overline{a}\langle x\rangle \mid a(x).\overline{b}\langle x\rangle \\ &\quad\xrightarrow[2.2]{} \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow a(x).\overline{b}\langle x\rangle \end{aligned}$$

These are all of the possible reductions. Note that none of the expressions in the reduction pathways can be expressed in the form $(\nu\, \tilde{c})(\overline{b}\langle \tilde{v}\rangle \mid R)$ where $b \notin \tilde{c}$, and so $C[P]' \not\downarrow_b$ for any process $C[P]'$ such that $C[P] \longrightarrow^* C[P]'$. Thus $C[P] \not\Downarrow_b$.
Now consider the other process:

$$\begin{aligned} C[P'] &= a(x).(\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\longrightarrow \overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\longrightarrow \overline{a}\langle x\rangle \mid a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\longrightarrow \overline{b}\langle x\rangle = C[P']' \end{aligned}$$

These are not all of the possible reductions; however, we reach a process $C[P']' = \overline{b}\langle x\rangle$ for which clearly $C[P']' \downarrow_b$. Thus we have that $C[P'] \longrightarrow^* C[P']' \downarrow_b$ and thus $C[P'] \Downarrow_b$.

Hence, $C[P]$ and $C[P']$ have different weak barbs and are thus differentiated by the context C. This context is of minimal size, since we require

1. One symbol $(-)$ to put the expression into
2. One output $\overline{a}\langle x\rangle$ to remove the input $a(x)$ from both processes
3. A new name b that is used for output, as both processes can output on a. This must be nested in two input expressions $a(x).a(x)$ as both processes can reduce at least once using an output $\overline{a}\langle x\rangle$, while only process P' can do this twice.

Hence the minimum size of context C that differentiates P and P' is

$$\begin{aligned}&\text{size}(- \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle) = \\ =&\,\text{size}(-) + \text{size}(\overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle)) = \\ =&\,1 + \text{size}(\overline{a}\langle x\rangle) + \text{size}(a(x).a(x).\overline{b}\langle x\rangle) = \\ =&\,1+1+3=5\end{aligned}$$

(3) Q and Q' are **not** reduction congruent. Proof:
Again, the context $C = - \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle$ separates Q and Q'. For process Q:

$$\begin{aligned}C[Q] &= !\,a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\xrightarrow[1]{} !\,a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid a(x).\overline{b}\langle x\rangle \\ &\equiv_2 !\,a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\quad\xrightarrow[2.1]{} !\,a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid a(x).\overline{b}\langle x\rangle \\ &\quad\xrightarrow[2.2]{} !\,a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\qquad\xrightarrow[2.2.1]{} !\,a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid a(x).\overline{a}\langle x\rangle \mid a(x).\overline{b}\langle x\rangle \\ &\qquad\xrightarrow[2.2.2]{} !\,a(x).(a(x).\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle = C[Q]\end{aligned}$$

These are all the possible reduction paths, as expansion and reduction of the replication does not give us any extra reductions in this case. Note that now, we either terminate or get back to the original process $C[Q]$. None of the processes $C[Q]'$ that $C[Q]$ can reduce to can be expressed in the form $(\nu\,\tilde{c})(\overline{b}\langle\tilde{v}\rangle \mid R)$ where $b \notin \tilde{c}$, and so $C[Q]' \not\downarrow_b$ for any process $C[Q]'$ such that $C[Q] \longrightarrow^* C[Q]'$. Thus $C[Q] \not\Downarrow_b$.
On the other hand, consider the following reductions for $C[Q']$:

$$\begin{aligned}C[Q'] &= !\,a(x).(\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\longrightarrow !\,a(x).(\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid a(x).(\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\longrightarrow !\,a(x).(\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid \overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle \mid a(x).a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\longrightarrow !\,a(x).(\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid \overline{a}\langle x\rangle \mid a(x).\overline{b}\langle x\rangle \longrightarrow \\ &\longrightarrow !\,a(x).(\overline{a}\langle x\rangle \mid \overline{a}\langle x\rangle) \mid \overline{b}\langle x\rangle = C[Q']'\end{aligned}$$

We observe that $C[Q']' = \overline{b}\langle x\rangle|R$ for some process R, which means that $C[Q']' \downarrow_b$ and thus $C[Q'] \Downarrow_b$. Hence the weak barbs of $C[Q]$ and $C[Q']$ are different, and therefore C differentiates the two processes. Again, C is of minimal size for the same reasons as in part (2) and thus the minimal size of a separating context is 5.

(c) Assume two statements:

(i) if $P \downarrow_a$ for some a, then $Q \downarrow_a$; and if $Q \downarrow_a$ for some a, then $P \downarrow_a$

(ii) if $P \Downarrow_a$ for some a, then $Q \Downarrow_a$; and if $Q \Downarrow_a$ for some a, then $P \Downarrow_a$

Does (i) imply (ii)? Does (ii) imply (i)?

Both implications are **false**.

Counterexample for (i) $\Longrightarrow$ (ii):

Let $P = \overline{b}\langle x\rangle$ and $Q = \overline{b}\langle a\rangle \mid b(x).\overline{x}\langle c\rangle$. Then $P \downarrow_b$ and $Q \downarrow_b$ with no other strong barbs; thus (i) holds. Now, while P cannot be reduced, $Q \longrightarrow \overline{a}\langle c\rangle$. So $Q \Downarrow_a$ while $P \not\Downarrow_a$. Hence (ii) does not hold and the implication (i) $\Longrightarrow$ (ii) is false. □

Counterexample for (ii) $\Longrightarrow$ (i):

First, recall that $P \Downarrow_a \iff P \longrightarrow^* P'$ and $P' \downarrow_a$. Since $P \longrightarrow^* P$ for all P, $P \downarrow_a \implies P \Downarrow_a$.

Let $P = \overline{a}\langle b\rangle \mid a(x).\overline{x}\langle c\rangle$ and $Q = \overline{b}\langle a\rangle \mid b(x).\overline{x}\langle c\rangle$. Clearly $P \downarrow_a$ and $Q \downarrow_b$, so also $P \Downarrow_a$ and $Q \Downarrow_b$. Now, $P \longrightarrow \overline{b}\langle c\rangle \downarrow_b$ so $P \Downarrow_b$ and $Q \longrightarrow \overline{a}\langle c\rangle \downarrow_a$ so $Q \Downarrow_a$. Thus the processes have the same weak barbs and (ii) holds. But then observe that $P \not\downarrow_b$ and $Q \not\downarrow_a$. Hence (i) does not hold and the implication (ii) $\Longrightarrow$ (i) is false. □

Question 2

(a) Explain the reason (with an example) why the polyadic π-calculus must be typed to prove the above encoding satisfies the criteria.

The polyadic π-calculus must be typed, as otherwise **completeness** does not hold. Proof:
Let $P = a(x_1, x_2).\mathbf{0} \mid \overline{a}\langle x\rangle.\mathbf{0}$ be an untyped polyadic π-calculus formula. Then

$$\begin{aligned}
[\![P]\!] &= a(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x_1).(\overline{y}\langle d\rangle \mid d(x_2).\mathbf{0})) \mid (\nu\, c)(\overline{a}\langle c\rangle \mid c(y).(\overline{y}\langle x\rangle \mid \mathbf{0})) \equiv \\
&\equiv (\nu\, c)(a(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x_1).(\overline{y}\langle d\rangle \mid d(x_2).\mathbf{0})) \mid \overline{a}\langle c\rangle \mid c(y).(\overline{y}\langle x\rangle \mid \mathbf{0})) \longrightarrow \\
&\longrightarrow (\nu\, c)((\nu\, d)(\overline{c}\langle d\rangle \mid d(x_1).(\overline{c}\langle d\rangle \mid d(x_2).\mathbf{0})) \mid c(y).(\overline{y}\langle x\rangle \mid \mathbf{0})) \equiv \\
&\equiv (\nu\, c, d)(\overline{c}\langle d\rangle \mid d(x_1).(\overline{c}\langle d\rangle \mid d(x_2).\mathbf{0}) \mid c(y).(\overline{y}\langle x\rangle \mid \mathbf{0})) = R
\end{aligned}$$

For completeness to hold, we would need to find a Q such that $P \longrightarrow Q$ and an R' such that $R \longrightarrow^* R'$ and $R' \cong [\![Q]\!]$. However, $P \not\longrightarrow$ as the identifier a sends and receives vectors of different arities. Thus completeness does not hold. □

(b) Assuming the polyadic π-calculus is typed, prove the above encoding satisfies the encodability criteria.

Let us first fully define the encoding recursively:

$$\begin{aligned}
[\![0]\!] &= 0 \\
[\![P \mid Q]\!] &= [\![P]\!] \mid [\![Q]\!] \\
[\![(\nu\, a)P]\!] &= (\nu\, a)[\![P]\!] \\
[\![!\, P]\!] &= !\, [\![P]\!] \\
[\![u(x_1, ..., x_n).P]\!] &= u(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x_1).(\overline{y}\langle d\rangle \mid d(x_2).(...(\overline{y}\langle d\rangle \mid d(x_n).[\![P]\!])))) \\
[\![\overline{u}\langle v_1, ..., v_n\rangle.P]\!] &= (\nu\, c)(\overline{u}\langle c\rangle \mid c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P]\!]))))
\end{aligned}$$

For this encoding, we must now prove 6 statements, which we will prove in this order:

1. Compositionality (homomorphic)
2. Closed under bijectional substitution
3. Soundness
4. Completeness
5. Barb preservation
6. Non-termination preservation

(1) Compositionality (homomorphic)

This statement requires the following four conditions to be true:

$$\begin{aligned}
[\![0]\!] &= 0 \\
[\![P \mid Q]\!] &= [\![P]\!] \mid [\![Q]\!] \\
[\![(\nu\, a)P]\!] &= (\nu\, a)[\![P]\!] \\
[\![!\, P]\!] &= !\, [\![P]\!]
\end{aligned}$$

These are true by our above definition. □

(2) Closed under bijectional substitution

Suppose that σ is a renaming for P. Let us prove the statement by induction on the structure of P.

Base case: if $P = \mathbf{0}$, then $\sigma' = \sigma$ works as $[\![\mathbf{0}\sigma]\!] = \mathbf{0} = [\![\mathbf{0}]\!]\sigma'$.

Inductive hypothesis: Suppose P_1 and P_2 are such that the statement holds: for each i, if σ_i is a renaming of P_i then there exists a renaming σ_i' such that $[\![P_i\sigma_i]\!] = [\![P_i]\!]\sigma_i'$.

Inductive steps:

Let us prove the statement for each of the 5 possible structures of P.

1. If $P = (\nu\, a)P_1$, then $[\![P\sigma]\!] = [\![((\nu\, a)P_1)\sigma]\!] = [\![(\nu\, a)P_1\sigma]\!] = (\nu\, a)[\![P_1\sigma]\!] = (\nu\, a)[\![P_1]\!]\sigma_1' = [\![(\nu\, a)P_1]\!]\sigma_1' = [\![P]\!]\sigma_1'$. Taking $\sigma' = \sigma_1'$ yields the result.

2. If $P = !\, P_1$, then $[\![P\sigma]\!] = [\![!\, P_1\sigma]\!] = [\![!\, P_1\sigma]\!] = !\, [\![P_1\sigma]\!] = !\, [\![P_1]\!]\sigma_1' = [\![!\, P_1]\!]\sigma_1' = [\![P]\!]\sigma_1'$. Taking $\sigma' = \sigma_1'$ yields the result.

3. If $P = u(x_1, ..., x_n).P_1$, then $[\![P\sigma]\!] = [\![u(x_1, ..., x_n)\sigma.P_1\sigma]\!]$. For the term $u(x_1, ..., x_n)\sigma$, either u is left unchanged by σ or it is renamed to some free name w.

   - In the former case, $[\![u(x_1, ..., x_n)\sigma.P_1\sigma]\!] =$
     $= [\![u(x_1, ..., x_n).P_1\sigma]\!] = u(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x_1).(\overline{y}\langle d\rangle \mid d(x_2).(...(\overline{y}\langle d\rangle \mid d(x_n).[\![P_1\sigma]\!])))) =$
     $= u(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x_1).(\overline{y}\langle d\rangle \mid d(x_2).(...(\overline{y}\langle d\rangle \mid d(x_n).[\![P_1]\!]\sigma_1')))) = [\![P]\!]\sigma_1'$. The last step holds as there are no other free names in the expression that σ' could rename. Thus taking $\sigma' = \sigma_1'$ yields the result.
   - In the latter case, $[\![u(x_1, ..., x_n)\sigma.P_1\sigma]\!] = [\![w(x_1, ..., x_n).P_1\sigma]\!] =$
     $= w(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x_1).(\overline{y}\langle d\rangle \mid d(x_2).(...(\overline{y}\langle d\rangle \mid d(x_n).[\![P_1\sigma]\!])))) =$
     $= w(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x_1).(\overline{y}\langle d\rangle \mid d(x_2).(...(\overline{y}\langle d\rangle \mid d(x_n).[\![P_1]\!]\sigma_1')))) = [\![P]\!]\sigma_1'\{w/u\}$, so taking $\sigma' = \sigma_1'\{w/u\}$ yields the result.

4. If $P = \overline{u}\langle v_1, ..., v_n\rangle.P_1$, then $[\![P\sigma]\!] = [\![\overline{u}\langle v_1, ..., v_n\rangle\sigma.P_1\sigma]\!]$. Again, we consider 2 cases: either u left unchanged by σ or it is renamed to some free name w.

   - In the former case, $[\![\overline{u}\langle v_1, ..., v_n\rangle\sigma.P_1\sigma]\!] = [\![\overline{u}\langle v_1, ..., v_n\rangle.P_1\sigma]\!] =$
     $= (\nu\, c)(\overline{u}\langle c\rangle \mid c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1\sigma]\!])))) =$
     $= (\nu\, c)(\overline{u}\langle c\rangle \mid c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!]\sigma_1')))) = [\![P]\!]\sigma_1'$. Hence taking $\sigma' = \sigma_1'$ yields the result.
   - In the latter case, $[\![\overline{u}\langle v_1, ..., v_n\rangle\sigma.P_1\sigma]\!] = [\![\overline{w}\langle v_1, ..., v_n\rangle.P_1\sigma]\!] =$
     $= (\nu\, c)(\overline{w}\langle c\rangle \mid c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1\sigma]\!])))) =$
     $= (\nu\, c)(\overline{w}\langle c\rangle \mid c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!]\sigma_1')))) = [\![P]\!]\sigma_1'\{w/u\}$. Hence taking $\sigma' = \sigma_1'\{w/u\}$ yields the result.

5. Finally, if $P = P_1 \mid P_2$, then $[\![P\sigma]\!] = [\![P_1\sigma \mid P_2\sigma]\!] = [\![P_1\sigma]\!] \mid [\![P_2\sigma]\!] = [\![P_1]\!]\sigma'_1 \mid [\![P_2]\!]\sigma'_2$. We can now merge the two renamings, σ'_1 and σ'_2 as $\sigma' = \sigma'_1\sigma'_2$. One situation where these two renamings could conflict would be if some free name a got renamed to some free name b by σ'_1 and to some other free name c by σ'_2. However, this is impossible, as in all of our other inductive steps the renaming has been systematically based on the renaming σ: a free name a got renamed to a free name b only if $\{b/a\}$ was in the renaming σ. Since σ cannot rename the same element to two different elements, the renamings σ'_1 and σ'_2 will not clash. The only other case when a conflict could occur is if σ'_1 renamed a free name a to b and σ'_2 left it alone (or vice versa). However, this can also not happen, as our previous inductive rules always rename a free name a to b if $\{b/a\}$ is in σ and a appears in the expression. Thus the two renamings can be merged without issue. Hence $[\![P_1]\!]\sigma'_1 \mid [\![P_2]\!]\sigma'_2 = [\![P_1]\!]\sigma' \mid [\![P_2]\!]\sigma' = ([\![P_1]\!] \mid [\![P_2]\!])\sigma' = [\![P]\!]\sigma'$.

Thus, by induction on the structure of P, the statement holds. □

(3) Soundness

Suppose that $P \longrightarrow Q$. We must show that there exists an R such that $[\![P]\!] \longrightarrow^* R$ and $R \cong [\![Q]\!]$.

Let us do this by induction on the possible reductions of P.

Induction hypothesis: Suppose that for some P_1, P_2 soundness holds: if $P_i \longrightarrow P_i'$, then $[\![P_i]\!] \longrightarrow^* R_i$ and $R_i \cong [\![P_i']\!]$.

Inductive steps:

Let us prove the statement for each of the 4 possible reduction steps of P.

1. If $P = P_1 \mid P_2$ and $P \longrightarrow P_1' \mid P_2 = Q$, then $[\![P]\!] = [\![P_1]\!] \mid [\![P_2]\!] \longrightarrow^* R_1 \mid [\![P_2]\!] = R$ and $R \cong [\![P_1']\!] \mid [\![P_2]\!] = [\![Q]\!]$.

2. If $P = (\nu\, a)P_1$ and $P \longrightarrow (\nu\, a)P_1' = Q$, then $[\![P]\!] = (\nu\, a)[\![P_1]\!] \longrightarrow^* (\nu\, a)R_1 = R$ and $R \cong (\nu\, a)[\![P_1']\!] = [\![Q]\!]$.

3. If $P \equiv P_1 \longrightarrow P_1' \equiv Q$, then we know that (by induction on the structural congruence) $[\![P]\!] \cong [\![P_1]\!]$. By definition of reduction congruence, $[\![P_1]\!] \longrightarrow R_1 \implies [\![P]\!] \longrightarrow^* R$ such that $R \cong R_1$. Since $R_1 \cong [\![Q]\!]$ and reduction congruence is an equivalence relation, we conclude that $R \cong [\![Q]\!]$.

4. If $P = \overline{u}\langle v_1, ..., v_n\rangle.P_1 \mid u(x_1, ..., x_n).P_2$ and $P \longrightarrow P_1 \mid P_2\{v_1, ..., v_n/x_1, ..., x_n\} = Q$, then

$$
\begin{aligned}
[\![P]\!] &= (\nu\, c)(\overline{u}\langle c\rangle \mid c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!])))) \mid \\
&\quad \mid u(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x_1).(\overline{y}\langle d\rangle \mid d(x_2).(...(\overline{y}\langle d\rangle \mid d(x_n).[\![P_2]\!])))) \longrightarrow \\
&\longrightarrow (\nu\, c, d)(c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!]))) \mid \\
&\quad \mid \overline{c}\langle d\rangle \mid d(x_1).(\overline{c}\langle d\rangle \mid d(x_2).(...(\overline{c}\langle d\rangle \mid d(x_n).[\![P_2]\!])))) \longrightarrow \\
&\longrightarrow (\nu\, c, d)(\overline{d}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!])) \mid \\
&\quad \mid d(x_1).(\overline{c}\langle d\rangle \mid d(x_2).(...(\overline{c}\langle d\rangle \mid d(x_n).[\![P_2]\!])))) \longrightarrow \\
&\longrightarrow (\nu\, c, d)(c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!])) \mid \\
&\quad \mid \overline{c}\langle d\rangle \mid d(x_2).(...(\overline{c}\langle d\rangle \mid d(x_n).[\![P_2\{v_1/x_1\}]\!]))) \longrightarrow \\
&\longrightarrow (\nu\, c, d)(\overline{d}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!]) \mid d(x_2).(...(\overline{c}\langle d\rangle \mid d(x_n).[\![P_2\{v_1/x_1\}]\!]))) \longrightarrow \\
&\longrightarrow (\nu\, c, d)(... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!]) \mid ...(\overline{c}\langle d\rangle \mid d(x_n).[\![P_2\{v_1, v_2/x_1, x_2\}]\!])) \longrightarrow \\
&\longrightarrow ... \longrightarrow \\
&\longrightarrow (\nu\, c, d)([\![P_1]\!] \mid [\![P_2]\!]\{v_1, ..., v_n/x_1, ..., x_n\}) \equiv [\![P_1]\!] \mid [\![P_2]\!]\{v_1, ..., v_n/x_1, ..., x_n\} = R
\end{aligned}
$$

   Note: to get to R, we used $2n + 1$ reduction steps. Now, $R \cong [\![Q]\!]$ and we are done.

Thus by induction on the reductions of P, soundness holds. □

(4) Completeness

In order to prove completeness, we will need to do induction on the reductions of the encoding. However, the same approach as for soundness will not work, as there does not always exist a reduction of the original process for a single reduction of the encoded process. Thus, we will aim to show that once one reduction step has been taken by the encoding, all future steps (up to a certain point) are uniquely determined and we are able to find a path that satisfies the criterion.

Suppose $[\![P]\!] \longrightarrow R$. We must show that $P \longrightarrow Q$ and $R \longrightarrow^* R'$ and $[\![Q]\!] \cong R'$.

Let us proceed by induction on the reductions of $[\![P]\!]$.

Inductive hypothesis: Suppose that for P_1, P_2 we have that if $[\![P_i]\!] \longrightarrow R_i$ then $P_i \longrightarrow Q_i$ and $R_i \longrightarrow^* R_i'$ and $[\![Q_i]\!] \cong R_i'$.

Inductive steps:

Let us prove the statement for each of the 4 possible reductions of $[\![P]\!]$.

1. Suppose $[\![P]\!] = P_3 \,|\, P_4$ and $P_3 \,|\, P_4 \longrightarrow P_3' \,|\, P_4 = R$. Since P_3 could reduce alone, we know from the encoding rules that it must have been encoded from some process $[\![P_1]\!] = P_3$. Hence also there exists some P_2 such that $[\![P_2]\!] = P_4$. Thus $[\![P]\!] = [\![P_1]\!] \mid [\![P_2]\!] = [\![P_1 \mid P_2]\!]$ and thus $P = P_1 \mid P_2$. From the induction hypothesis, $[\![P_1]\!] = P_3 \longrightarrow P_3' = R_1$, $P_1 \longrightarrow Q_1$, $R_1 \longrightarrow^* R_1'$, $[\![Q_1]\!] \cong R_1'$. Then $P = P_1 \,|\, P_2 \longrightarrow Q_1 \,|\, P_2 = Q$, and $[\![P]\!] \longrightarrow P_3' \,|\, P_4 = R_1 \,|\, P_4 = R$, and $R \longrightarrow^* R_1' \,|\, R_4 = R'$. Since $[\![Q_1]\!] \cong R_1'$, we conclude that $[\![Q]\!] = [\![Q_1 \mid P_2]\!] = [\![Q_1]\!] \mid P_4 \cong R_1' \mid P_4 = R'$.

2. Suppose $[\![P]\!] = (\nu\, a)P_3$ and $(\nu\, a)P_3 \longrightarrow (\nu\, a)P_3' = R$. Then we have two possible cases: either $P_3 = [\![P_1]\!]$ for some P_1 or it is not.

   - In the former case, $[\![P]\!] = (\nu\, a)[\![P_1]\!] \implies P = (\nu\, a)P_1$. From the induction hypothesis, $[\![P_1]\!] \longrightarrow P_3' = R_1$, $P_1 \longrightarrow Q_1$, $R_1 \longrightarrow^* R_1'$, $[\![Q_1]\!] \cong R_1'$. Then $P = (\nu\, a)P_1 \longrightarrow (\nu\, a)Q_1 = Q_1$ and $R = (\nu\, a)R_1 \longrightarrow^* (\nu\, a)R_1'$. We conclude that $[\![Q]\!] = (\nu\, a)[\![Q_1]\!] \cong (\nu\, a)R_1' = R'$.
   - In the latter case, we know that the reduction is the first step in the reduction of an expression $P = \overline{u}\langle v_1, ..., v_n\rangle.P_1 \mid u(x_1, ..., x_n).P_2$ for which

$$\begin{aligned}
[\![P]\!] &= (\nu\, c)(\overline{u}\langle c\rangle \mid c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!])))) \mid \\
&\quad \mid u(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x_1).(\overline{y}\langle d\rangle \mid d(x_2).(...(\overline{y}\langle d\rangle \mid d(x_n).[\![P_2]\!])))) \equiv \\
&\equiv (\nu\, c)(\overline{u}\langle c\rangle \mid c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!]))) \mid \\
&\quad \mid u(y).(\nu\, d)(\overline{y}\langle d\rangle \mid d(x_1).(\overline{y}\langle d\rangle \mid d(x_2).(...(\overline{y}\langle d\rangle \mid d(x_n).[\![P_2]\!]))))) \longrightarrow \\
&\longrightarrow (\nu\, c)(c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!]))) \mid \\
&\quad \mid (\nu\, d)(\overline{c}\langle d\rangle \mid d(x_1).(\overline{c}\langle d\rangle \mid d(x_2).(...(\overline{c}\langle d\rangle \mid d(x_n).[\![P_2]\!]))))) = \\
&= R
\end{aligned}$$

     We have already evaluated this expression in our proof of Soundness, and have seen that it takes $2n$ more steps to reduce to the expression $[\![P_1]\!] \,|\, [\![P_2]\!]\{v_1, ..., v_n/x_1, ..., x_n\} = R'$. Hence $P \longrightarrow P_1 \mid P_2\{v_1, ..., v_n/x_1, ..., x_n\} = Q$, and $[\![Q]\!] \cong R'$.

3. The case $[\![P]\!] = u(x).P_1 \mid \overline{u}\langle v\rangle$ cannot happen, as none of our encoding rules encode a single process in this way. Output always gets encoded using a restriction, which is covered in the previous case.

4. Suppose $[\![P]\!] \equiv P_3 \longrightarrow P_3' \equiv R$. Then there exists P_1 such that $[\![P_1]\!] = P_3$ and $P \equiv P_1$. Then by the induction hypothesis $[\![P_1]\!] = P_3 \longrightarrow P_3' = R_1 = R$, $P_1 \longrightarrow Q_1$, $R_1 \longrightarrow^* R_1'$, $[\![Q_1]\!] \cong R_1' = R'$. So also $P \longrightarrow Q_1 = Q$ with $[\![Q_1]\!] \cong R$.

Thus by induction on the reductions of $[\![P]\!]$, completeness holds. □

(5) Barb preservation

($P \Downarrow_a \implies [\![P]\!] \Downarrow_a$)

Suppose $P \Downarrow_a$ for some a. Then either $P \downarrow_a$ or $P \longrightarrow^* Q$ and $Q \downarrow_a$ (in the latter case, we assume WLOG that the reduction takes more than zero steps).

- If $P \downarrow_a$, then $P = \overline{a}\langle v_1, ..., v_n\rangle \mid P'$ for some process P'. Then $[\![P]\!] = (\nu\, c)(\overline{a}\langle c\rangle \mid c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!])))) \mid [\![P']\!]$ which clearly has $[\![P]\!] \downarrow_a$ and thus $[\![P]\!] \Downarrow_a$.

- If $P \longrightarrow^* Q$ and $Q \downarrow_a$, we can show that $[\![P]\!] \Downarrow_a$ by induction on reductions. Suppose $P_1 \longrightarrow P_2$ in one step, $P_1 \Downarrow_a$ and for P_2 the statement $P_2 \Downarrow_a \implies [\![P_2]\!] \Downarrow_a$ holds. Then by soundness, $[\![P_1]\!] \longrightarrow^* R$, $R \cong [\![P_2]\!]$. By the induction hypothesis $[\![P_2]\!] \Downarrow_a$ which implies that $R \Downarrow_a$ and therefore $[\![P_1]\!] \Downarrow_a$.

  Since $Q \downarrow_a$, we have a base case and therefore by induction $[\![P]\!] \Downarrow_a$.

($[\![P]\!] \Downarrow_a \implies P \Downarrow_a$)

Suppose $[\![P]\!] \Downarrow_a$ for some a. Then either $[\![P]\!] \downarrow_a$ or $[\![P]\!] \longrightarrow^* Q$ and $Q \downarrow_a$ (again, in the latter case, we assume WLOG that the reduction takes more than zero steps).

- If $[\![P]\!] \downarrow_a$, then we must have that $P = \overline{a}\langle v_1, ..., v_n\rangle \mid P'$ for some process P' with $[\![P]\!] = (\nu\, c)(\overline{a}\langle c\rangle \mid c(y_1).(\overline{y_1}\langle v_1\rangle \mid c(y_2).(\overline{y_2}\langle v_2\rangle \mid ... \mid c(y_n).(\overline{y_n}\langle v_n\rangle \mid [\![P_1]\!])))) \mid [\![P']\!]$ as this is the only way for $[\![P]\!]$ to have a strong barb of a.

- If $[\![P]\!] \longrightarrow^* Q \downarrow_a$, we can show that $P \Downarrow_a$ by induction on reductions. Suppose $[\![P_1]\!] \longrightarrow P_2$ in one step, $[\![P_1]\!] \Downarrow_a$. Then by completeness, $P_1 \longrightarrow Q_1$, $P_2 \longrightarrow^* P_2'$, $P_2' \cong [\![Q_1]\!]$. By the induction hypothesis $[\![Q_1]\!] \Downarrow_a \implies Q_1 \Downarrow_a$ and thus also $P_1 \Downarrow_a$.

  Since $Q \downarrow_a$, we have a base case and therefore by induction $P \Downarrow_a$.

(6) Nontermination preservation

Let us define nontermination as follows: $P \Uparrow$ if and only if for all $s \in \mathbb{N}$, $P \longrightarrow^s$, where the notation $P \longrightarrow^s$ is defined as $P \longrightarrow P_2 \longrightarrow ... \longrightarrow P_s$ for some processes $\{P_i\}_{i=2}^s$. In other words, $P \longrightarrow^s$ if and only if P can take s reduction steps. Let us use this definition to prove the result in both directions.

($P \Uparrow \implies [\![P]\!] \Uparrow$)

Suppose $P \Uparrow$. Given any s, we must find a sequence of s reductions for $[\![P]\!]$. We know that $P \longrightarrow^s Q$ for some Q. Claim: for any possible reduction step of P, $[\![P]\!]$ will take one or more reduction steps. Let us prove this by induction on the possible reductions of P.

Suppose P_1, P_2 are such that $P_1 \longrightarrow P_1' \implies [\![P_1]\!] \longrightarrow^k [\![P_1]\!]'$, where $k \geq 1$. Consider every reduction P could make:

1. If $P = P_1 \mid P_2$ and $P_1 \mid P_2 \longrightarrow P_1' \mid P_2 = P'$, then $[\![P]\!] = [\![P_1]\!] \mid [\![P_2]\!] \longrightarrow^k [\![P_1]\!]' \mid [\![P_2]\!]$.
2. If $P = (\nu\, a)P_1$ and $(\nu\, a)P_1 \longrightarrow (\nu\, a)P_1'$, then $[\![P]\!] = (\nu\, a)[\![P_1]\!] \longrightarrow^k (\nu\, a)[\![P_1]\!]'$.
3. If $P \equiv P_1 \longrightarrow P_1' \equiv P'$ then we know directly that $P \longrightarrow P'$ and $[\![P]\!] = [\![P_1]\!] \longrightarrow^k [\![P_1]\!]'$
4. If $P = \overline{u}\langle v_1, ..., v_n\rangle.P_1 \mid u(x_1, ..., x_n).P_2$ then $P \longrightarrow P_1 \mid P_2\{v_1, ..., v_n/x_1, ..., x_n\}$. We have seen already that the reduction of the encoding takes $2n+1$ steps to reach a congruent process, which is always more steps than 1.

Hence for all reduction steps of P, $[\![P]\!]$ has to take more reduction steps and thus can do at least s steps for any reduction path of length s of the process P. Therefore $[\![P]\!] \Uparrow$.

($[\![P]\!] \Uparrow \implies P \Uparrow$)

Suppose $[\![P]\!] \Uparrow$. Given any s, we must find a sequence of s reductions for P. We know that $[\![P]\!] \longrightarrow^t Q$ for any t. Let $t = s(2n + 1)$. We know from the proof of the other direction that the longest reduction chain of $[\![P]\!]$ for one reduction step of P is of length $2n+1$, which is achieved for any process $P = \overline{u}\langle v_1, ..., v_n\rangle.P_1 \mid u(x_1, ..., x_n).P_2$. In the worst case, P will be fully composed of these, and $[\![P]\!]$ will have to take $s(2n+1)$ reduction steps to allow P to reduce s steps. It is always possible to find a reduction of length t for $[\![P]\!]$ and thus $P \Uparrow$.

We have proven the statement in both directions and therefore nontermination preservation holds. □

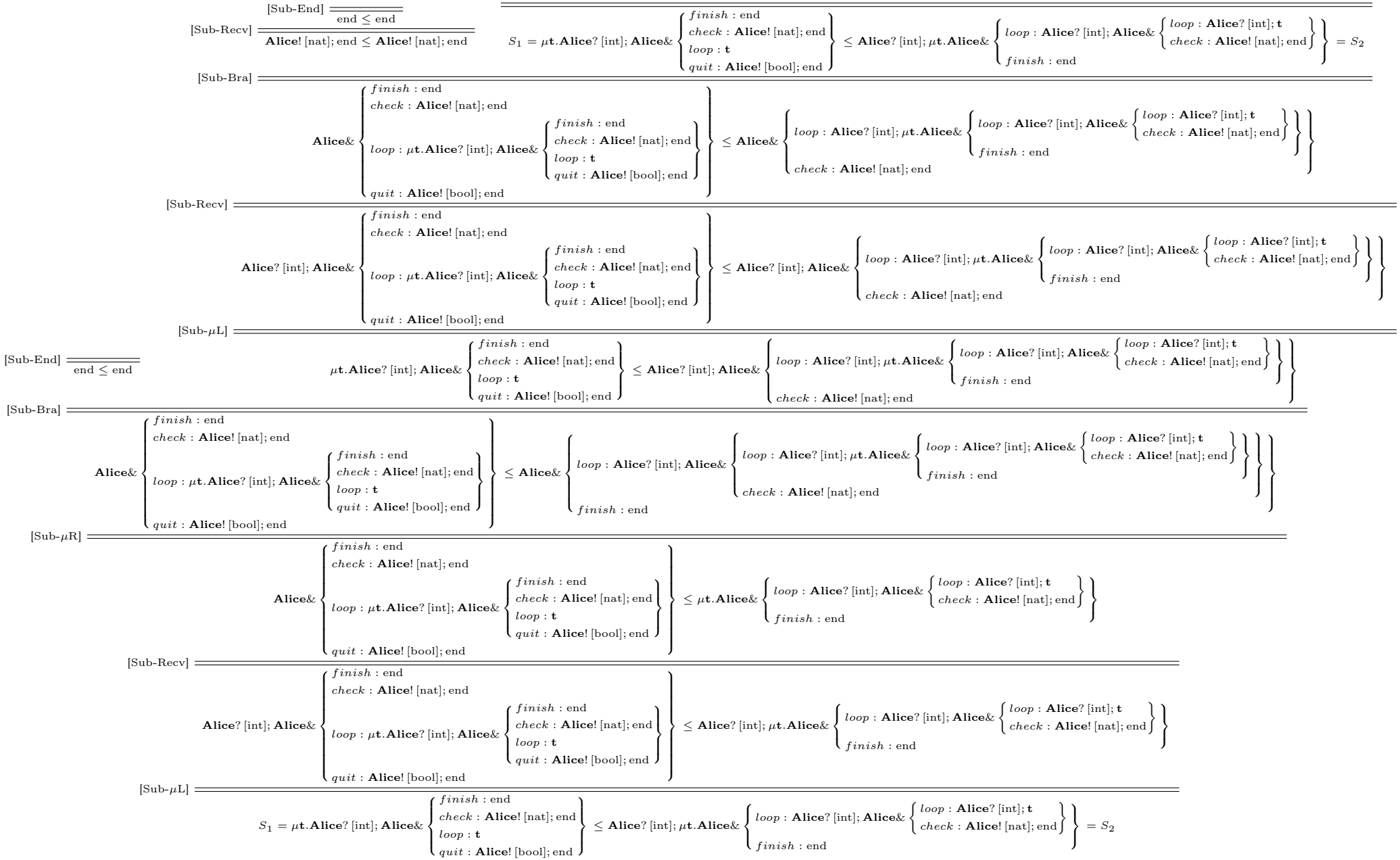
Question 3

(a) Assume the following session types, S_1 and S_2. Do they satisfy $S_1 \leq S_2$?

$$S_1 = \mu \mathbf{t}.\mathbf{Alice}?\,[\text{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \text{end} \\ check : \mathbf{Alice!}\,[\text{nat}];\text{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice!}\,[\text{bool}];\text{end} \end{array}\right\}$$

$$S_2 = \mathbf{Alice}?\,[\text{int}];\mu \mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\text{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\text{int}];\mathbf{t} \\ check : \mathbf{Alice!}\,[\text{nat}];\text{end} \end{array}\right\} \\ finish : \text{end} \end{array}\right\}$$

The session types S_1 and S_2 **do satisfy** $S_1 \leq S_2$. Let us first draw a derivation tree, after which we can encode it as a relation as in Solution V3. The derivation tree can be found on the following page.


[Sub-End] end ≤ end
[Sub-Recv] Alice! [nat]; end ≤ Alice! [nat]; end
$S_1 = \mu\mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \leq \mathbf{Alice}?\,[\mathrm{int}];\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\} = S_2$
[Sub-Bra]
$\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mu\mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \leq \mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\}$
[Sub-Recv]
$\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mu\mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \leq \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\}$
[Sub-μL]
[Sub-End] end ≤ end
$\mu\mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \leq \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\}$
[Sub-Bra]
$\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mu\mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \leq \mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\}$
[Sub-μR]
$\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mu\mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \leq \mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\}$
[Sub-Recv]
$\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mu\mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \leq \mathbf{Alice}?\,[\mathrm{int}];\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\}$
[Sub-μL]
$S_1 = \mu\mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \leq \mathbf{Alice}?\,[\mathrm{int}];\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\} = S_2$

Let us now package this tree as a subtyping relation $\mathcal{R}$. Let $\mathcal{S}$ be a set of session types defined as

$$
\mathcal{S} = \left\{
\begin{array}{l}
\mu\mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\},\\
\mathbf{Alice}?\,[\mathrm{int}];\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\},\\
\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mu\mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end}\end{array}\right\},\\
\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\},\\
\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\},\\
\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mu\mathbf{t}.\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{\begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\} \\ finish : \mathrm{end}\end{array}\right\} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}\end{array}\right\},\\
\mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end},\\
\mathrm{end}
\end{array}
\right\}
$$

Now, let us construct the subtyping relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ by encoding the information of the above derivation tree. First, we ensure that $(S_1, S_2) \in \mathcal{R}$. Then, for every inference rule in the tree which is not [Sub-μR] or [Sub-μL] we will add a pair (S_1', S_2') to the relation $\mathcal{R}$ where we take the session types S_1' and S_2' from the top of the inference rule, $S_1' \leq S_2'$. This results in the following relation:

$$\mathcal{R} = \left\{ \begin{array}{l} \left(\begin{array}{l} \mu \mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{ \begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end} \end{array} \right., \\ \mathbf{Alice}?\,[\mathrm{int}];\mu \mathbf{t}.\mathbf{Alice}\&\left\{ \begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{ \begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \end{array} \right\} \\ finish : \mathrm{end} \end{array} \right\} \end{array} \right), \\ \left(\begin{array}{l} \mathbf{Alice}\&\left\{ \begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mu \mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{ \begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end} \end{array} \right\} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end} \end{array} \right\}, \\ \mu \mathbf{t}.\mathbf{Alice}\&\left\{ \begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{ \begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \end{array} \right\} \\ finish : \mathrm{end} \end{array} \right\} \end{array} \right), \\ \left(\begin{array}{l} \mu \mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{ \begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end} \end{array} \right., \\ \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{ \begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mu \mathbf{t}.\mathbf{Alice}\&\left\{ \begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{ \begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \end{array} \right\} \\ finish : \mathrm{end} \end{array} \right\} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \end{array} \right\} \end{array} \right), \\ \left(\begin{array}{l} \mathbf{Alice}\&\left\{ \begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mu \mathbf{t}.\mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{ \begin{array}{l} finish : \mathrm{end} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \\ loop : \mathbf{t} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end} \end{array} \right\} \\ quit : \mathbf{Alice}!\,[\mathrm{bool}];\mathrm{end} \end{array} \right\}, \\ \mathbf{Alice}\&\left\{ \begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mu \mathbf{t}.\mathbf{Alice}\&\left\{ \begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{Alice}\&\left\{ \begin{array}{l} loop : \mathbf{Alice}?\,[\mathrm{int}];\mathbf{t} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \end{array} \right\} \\ finish : \mathrm{end} \end{array} \right\} \\ check : \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end} \end{array} \right\} \end{array} \right), \\ (\mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}, \mathbf{Alice}!\,[\mathrm{nat}];\mathrm{end}), \\ (\mathrm{end}, \mathrm{end}) \end{array} \right\}$$

By construction, the conditions of a subtyping hold as per Solution V3 from the slides. □

(b)

(1) Explain why the following global type is not well-formed under either plain or full merging:

$$G_{bad} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{B} \left\{ \begin{array}{l} ping : \mathbf{B} \to \mathbf{A} \left\{ pong : \mathbf{t} \right\} \\ stop : \text{end} \end{array} \right\} \\ reject : \mathbf{C} \to \mathbf{B} \left\{ ping : \mathbf{B} \to \mathbf{A} \left\{ stop : \text{end} \right\} \right\} \end{array} \right\}$$

(2) Amend G_{bad} to make it well-formed under full merging but not under plain merging. Give the projection of the global type you fixed onto **A**, **B**, and **C**.

(1) Let us try to project G_{bad} onto **C** and apply plain and full merging.

Firstly, since **C** is a participant inside the recursion, we have that

$$G_{bad} \upharpoonright \mathbf{C} = \left(\mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{B} \left\{ \begin{array}{l} ping : \mathbf{B} \to \mathbf{A} \left\{ pong : \mathbf{t} \right\} \\ stop : \text{end} \end{array} \right\} \\ reject : \mathbf{C} \to \mathbf{B} \left\{ ping : \mathbf{B} \to \mathbf{A} \left\{ stop : \text{end} \right\} \right\} \end{array} \right\} \right) \upharpoonright \mathbf{C} =$$

$$= \mu \mathbf{t}. \left(\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{B} \left\{ \begin{array}{l} ping : \mathbf{B} \to \mathbf{A} \left\{ pong : \mathbf{t} \right\} \\ stop : \text{end} \end{array} \right\} \\ reject : \mathbf{C} \to \mathbf{B} \left\{ ping : \mathbf{B} \to \mathbf{A} \left\{ stop : \text{end} \right\} \right\} \end{array} \right\} \right) \upharpoonright \mathbf{C}$$

Since **C** is different from both **A** and **B**, we must determine and merge $G_{accept} \upharpoonright \mathbf{C}$ and $G_{reject} \upharpoonright \mathbf{C}$,

$$G_{accept} = \mathbf{C} \to \mathbf{B} \left\{ \begin{array}{l} ping : \mathbf{B} \to \mathbf{A} \left\{ pong : \mathbf{t} \right\} \\ stop : \text{end} \end{array} \right\}$$

$$G_{reject} = \mathbf{C} \to \mathbf{B} \left\{ ping : \mathbf{B} \to \mathbf{A} \left\{ stop : \text{end} \right\} \right\}$$

By applying the rules again, we find that the two values are:

$$G_{accept} \upharpoonright \mathbf{C} = \mathbf{B} \oplus \left\{ \begin{array}{l} ping : \mathbf{t} \\ stop : \text{end} \end{array} \right\}$$

$$G_{reject} \upharpoonright \mathbf{C} = \mathbf{B} \oplus \left\{ ping : \text{end} \right\}$$

G_{bad} is not well-formed under plain merging, as $G_{accept} \neq G_{reject}$. G_{bad} is not well-formed under full merging either, since the rule for $G_{accept} \sqcap G_{reject}$ requires that both expressions have the same labels in the selection, which is not true. Furthermore, even if this requirement was dropped, the *ping* branch cannot be merged, as $\mathbf{t} \sqcap \text{end}$ is undefined. □

(2) Claim: the following global type, G_{good}, which is amended from the above G_{bad}, is well-formed under full merging but not under plain merging.

$$G_{good} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{B} \to \mathbf{C} \left\{ ping : \mathbf{B} \to \mathbf{A} \left\{ pong : \mathbf{t} \right\} \right\} \\ reject : \mathbf{B} \to \mathbf{C} \left\{ stop_1 : \mathbf{B} \to \mathbf{A} \left\{ stop_2 : \mathrm{end} \right\} \right\} \end{array} \right\}$$

Proof:

Let us project the global type onto all of the participants to see why this is the case.

$$\begin{aligned} G_{good} \upharpoonright \mathbf{C} &= \left(\mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{B} \to \mathbf{C} \left\{ ping : \mathbf{B} \to \mathbf{A} \left\{ pong : \mathbf{t} \right\} \right\} \\ reject : \mathbf{B} \to \mathbf{C} \left\{ stop_1 : \mathbf{B} \to \mathbf{A} \left\{ stop_2 : \mathrm{end} \right\} \right\} \end{array} \right\} \right) \upharpoonright \mathbf{C} = \\ &= \mu \mathbf{t}. \left(\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{B} \to \mathbf{C} \left\{ ping : \mathbf{B} \to \mathbf{A} \left\{ pong : \mathbf{t} \right\} \right\} \\ reject : \mathbf{B} \to \mathbf{C} \left\{ stop_1 : \mathbf{B} \to \mathbf{A} \left\{ stop_2 : \mathrm{end} \right\} \right\} \end{array} \right\} \right) \upharpoonright \mathbf{C} \end{aligned}$$

Since $\mathbf{C}$ is different from both $\mathbf{A}$ and $\mathbf{B}$, we must determine and merge $G_{accept} \upharpoonright \mathbf{C}$ and $G_{reject} \upharpoonright \mathbf{C}$.

$$\begin{aligned} G_{accept} &= \mathbf{B} \to \mathbf{C} \left\{ ping : \mathbf{B} \to \mathbf{A} \left\{ pong : \mathbf{t} \right\} \right\} \\ G_{reject} &= \mathbf{B} \to \mathbf{C} \left\{ stop_1 : \mathbf{B} \to \mathbf{A} \left\{ stop_2 : \mathrm{end} \right\} \right\} \\ G_{accept} \upharpoonright \mathbf{C} &= \mathbf{B} \& \left\{ ping : \mathbf{t} \right\} \\ G_{reject} \upharpoonright \mathbf{C} &= \mathbf{B} \& \left\{ stop_1 : \mathrm{end} \right\} \end{aligned}$$

The expressions $G_{accept} \upharpoonright \mathbf{C}$ and $G_{reject} \upharpoonright \mathbf{C}$ are not mergeable under plain merging, as they are different; thus, G_{good} is not well-formed under plain merging. However, they are mergeable under full merging as

$$(G_{accept} \upharpoonright \mathbf{C}) \sqcap (G_{reject} \upharpoonright \mathbf{C}) = \mathbf{B} \& \left\{ \begin{array}{l} ping : \mathbf{t} \\ stop_1 : \mathrm{end} \end{array} \right\}$$

By applying the projection rules to all participants, we get that

$$\begin{aligned} G_{good} \upharpoonright \mathbf{A} &= \mu \mathbf{t}.\mathbf{B} \oplus \left\{ \begin{array}{l} accept : \mathbf{B} \& \left\{ pong : \mathbf{t} \right\} \\ reject : \mathbf{B} \& \left\{ stop_2 : \mathrm{end} \right\} \end{array} \right\} \\ G_{good} \upharpoonright \mathbf{B} &= \mu \mathbf{t}.\mathbf{A} \& \left\{ \begin{array}{l} accept : \mathbf{C} \oplus \left\{ ping : \mathbf{A} \oplus \left\{ pong : \mathbf{t} \right\} \right\} \\ reject : \mathbf{C} \oplus \left\{ stop_1 : \mathbf{A} \oplus \left\{ stop_2 : \mathrm{end} \right\} \right\} \end{array} \right\} \\ G_{good} \upharpoonright \mathbf{C} &= \mu \mathbf{t}.\mathbf{B} \& \left\{ \begin{array}{l} ping : \mathbf{t} \\ stop_1 : \mathrm{end} \end{array} \right\} \end{aligned}$$

Thus G_{good} is well-formed under full merging with the above projections onto the participants. □

Question 4

(a) Explain with an example why the side condition "$\mu\mathbf{t}.G$ is closed" is required for the projection of the recursive type.

Recall that the definition of "closed": a session type G is closed if G does not contain any type variables $\mathbf{t}$ that have not been bound by an expression $\mu\mathbf{t}$. For example, the session type $\mu\mathbf{t}.\mathbf{A} \to \mathbf{B}\left\{\begin{array}{l} ping : \mathbf{t} \\ stop : \text{end} \end{array}\right\}$ is closed, as the only type variable $\mathbf{t}$ is bound.

However, consider the session type

$$G = \mu\mathbf{u}.\mathbf{A} \to \mathbf{C}[\text{int}]; \mu\mathbf{t}.\mathbf{A} \to \mathbf{B}\left\{\begin{array}{l} ping : \mathbf{t} \\ pong : \mathbf{u} \\ stop : \text{end} \end{array}\right\}$$

Let us project this type onto the participant $\mathbf{C}$.

$$G \upharpoonright \mathbf{C} = \left(\mu\mathbf{u}.\mathbf{A} \to \mathbf{C}[\text{int}]; \mu\mathbf{t}.\mathbf{A} \to \mathbf{B}\left\{\begin{array}{l} ping : \mathbf{t} \\ pong : \mathbf{u} \\ stop : \text{end} \end{array}\right\}\right) \upharpoonright \mathbf{C} =$$

$$= \mu\mathbf{u}.\mathbf{A}?\,[\text{int}]; \left(\mu\mathbf{t}.\mathbf{A} \to \mathbf{B}\left\{\begin{array}{l} ping : \mathbf{t} \\ pong : \mathbf{u} \\ stop : \text{end} \end{array}\right\}\right) \upharpoonright \mathbf{C}$$

Notice that the type inside the brackets is not closed, as it has a free occurrence of a type variable $\mathbf{u}$. Now, if the condition "$\mu\mathbf{t}.G$ is closed" did not exist, we would have to terminate the projection as $G \upharpoonright \mathbf{C} = \mu\mathbf{u}.\mathbf{A}?\,[\text{int}]; \text{end}$, which is a badly formed type as the $\mu\mathbf{u}.$ does not have a corresponding $\mathbf{u}$. Instead, given the correct projection rule, we conclude that G cannot be projected onto $\mathbf{C}$ as the different branches (*ping*, *pong*, *stop*) cannot be merged with the $\sqcap$ operator. Hence the rule is required so that applying our projection rules does not result in invalid session types.

(b) Consider the two following global types:

$$G_1 = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\}$$

$$G_2 = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\}$$

(1) Prove that they represent the same protocol using the transition relation $G \stackrel{\alpha}{\Longrightarrow} G'$ defined in *Global Type Semantics* in "Subject Reduction Proof" available from the course web page.

(2) G_1 and G_2 are not projectable under either plain or full merging given in the slides. Define the projection rules which enable to project G_1 and G_2, and justify your rules using G_1 and G_2.

(1)

Consider the following steps:

$$G_1 = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} =$$

$$= \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} =$$

$$= \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \\ reject : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \Rightarrow$$

$$\stackrel{CD:[\text{nat}]}{\Longrightarrow} \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} =$$

$$= \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \\ reject : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \stackrel{CD:[\text{nat}]}{\Longrightarrow}$$

$$\stackrel{CD:[\text{nat}]}{\Longrightarrow} \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} =$$

$$= \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : G_1 \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\}$$

$$G_2 = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} =$$

$$= \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} =$$

$$= \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \\ reject : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \xrightarrow{CD:[\text{nat}]}$$

$$\xrightarrow{CD:[\text{nat}]} \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t} \\ reject : \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} \\ reject : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\} =$$

$$= \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : G_2 \\ reject : \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}' \end{array} \right\}$$

Now, observe that in the steps of G_1, the *reject* branch is equal to $\mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}'$, while in the steps of G_2, it is equal to $\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mu \mathbf{t}'.\mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{C} \to \mathbf{D} : [\text{nat}]; \mathbf{t}'$. The only semantic rule we are able to apply for both of these types is $\xrightarrow{CD:[\text{nat}]}$ ad infinitum. Thus the global type semantics defined in the paper cannot distinguish between the two global types, and we can say that they are equivalent to a global type G_3. Therefore

$$G_1 \Longrightarrow^* \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : G_1 \\ reject : G_3 \end{array} \right\} = G_1'$$

$$G_2 \Longrightarrow^* \mathbf{A} \to \mathbf{B} \left\{ \begin{array}{l} accept : G_2 \\ reject : G_3 \end{array} \right\} = G_2'$$

where

$$G_1' \xrightarrow{AB:accept} G_1$$

$$G_1' \xrightarrow{AB:reject} G_3$$

$$G_2' \xrightarrow{AB:accept} G_2$$

$$G_2' \xrightarrow{AB:reject} G_3$$

Thus the global types cannot be distinguished by our defined global type semantics, as they admit the same transitions $\stackrel{\alpha}{\Longrightarrow}$, and thus they represent the same protocol.

(2)

Let us define adjust the projection rules to expand the definition of $\sqcap$. Suppose that for some global type G and participant A, we have that $G \upharpoonright A = G_1.\mu \mathbf{t}.G_2.\prod_{i \in I} G_i$, where G_i is open for some i with respect to $\mathbf{t}$. Then,

$$G \upharpoonright A = G_1.\prod_{i \in I} \begin{cases} G_i, & G_i \text{ is closed w.r.t. } \mathbf{t} \\ \mu \mathbf{t}.G_2.G_i, & G_i \text{ is open w.r.t. } \mathbf{t} \end{cases}$$

In addition, we need to add a rule for expanding recursions during merging. We do this by defining the following:

$$\mu \mathbf{t}.S \sqcap \mu \mathbf{t}.S' = \mu \mathbf{t}.S \sqcap \mu \mathbf{t}.S'\{S'/t\}$$
$$\mu \mathbf{t}.S \sqcap \mu \mathbf{t}.S' = \mu \mathbf{t}.S\{S/t\} \sqcap \mu \mathbf{t}.S'$$

with the additional remark that $\mu \mathbf{t}'.S = \mu \mathbf{t}.S\{t/t'\}$ if t does not appear anywhere in S.

Let us use these new rules to project G_1 and G_2 onto all participants. For the former,

$$\begin{aligned}
G_1 \upharpoonright A &= \mu \mathbf{t}.B \oplus \begin{cases} accept : \mathbf{t} \\ reject : \text{end} \end{cases} \\
G_1 \upharpoonright B &= \mu \mathbf{t}.A \& \begin{cases} accept : \mathbf{t} \\ reject : \text{end} \end{cases} \\
G_1 \upharpoonright C &= \mu \mathbf{t}.((D![\text{nat}]; D![\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}'.D![\text{nat}]; \mathbf{t}')) = \\
&= (\mu \mathbf{t}.D![\text{nat}]; D![\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}'.D![\text{nat}]; \mathbf{t}') = \\
&= (\mu \mathbf{t}.D![\text{nat}]; D![\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}.D![\text{nat}]; \mathbf{t}) = \\
&= (\mu \mathbf{t}.D![\text{nat}]; D![\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}.D![\text{nat}]; D![\text{nat}]; \mathbf{t}) = \\
&= \mu \mathbf{t}.D![\text{nat}]; D![\text{nat}]; \mathbf{t} \\
G_1 \upharpoonright D &= \mu \mathbf{t}.((C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}'.C?[\text{nat}]; \mathbf{t}')) = \\
&= (\mu \mathbf{t}.C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}'.C?[\text{nat}]; \mathbf{t}') = \\
&= (\mu \mathbf{t}.C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}.C?[\text{nat}]; \mathbf{t}) = \\
&= (\mu \mathbf{t}.C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}.C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}) = \\
&= \mu \mathbf{t}.C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}
\end{aligned}$$

For the latter,

$$\begin{aligned}
G_2 \upharpoonright A &= \mu \mathbf{t}.B \oplus \begin{cases} accept : \mathbf{t} \\ reject : \text{end} \end{cases} \\
G_2 \upharpoonright B &= \mu \mathbf{t}.A \& \begin{cases} accept : \mathbf{t} \\ reject : \text{end} \end{cases} \\
G_2 \upharpoonright C &= \mu \mathbf{t}.((D![\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}'.D![\text{nat}]; D![\text{nat}]; \mathbf{t}')) = \\
&= (\mu \mathbf{t}.D![\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}'.D![\text{nat}]; D![\text{nat}]; \mathbf{t}') = \\
&= (\mu \mathbf{t}.D![\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}.D![\text{nat}]; D![\text{nat}]; \mathbf{t}) = \\
&= (\mu \mathbf{t}.D![\text{nat}]; D![\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}.D![\text{nat}]; D![\text{nat}]; \mathbf{t}) = \\
&= \mu \mathbf{t}.D![\text{nat}]; D![\text{nat}]; \mathbf{t} \\
G_2 \upharpoonright D &= \mu \mathbf{t}.((C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}'.C?[\text{nat}]; \mathbf{t}')) = \\
&= (\mu \mathbf{t}.C?[\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}'.C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}') = \\
&= (\mu \mathbf{t}.C?[\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}.C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}) = \\
&= (\mu \mathbf{t}.C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}) \sqcap (\mu \mathbf{t}.C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}) = \\
&= \mu \mathbf{t}.C?[\text{nat}]; C?[\text{nat}]; \mathbf{t}
\end{aligned}$$

□