Linear Functions

Task 1 (L22.3, 20 points) Recall the definition of a purely positive type, updated to reflect the notation for linear types.

\[
\tau^+ ::= 1 \mid \tau^+_1 \otimes \tau^+_2 \mid \oplus_{i \in I}(i : \tau^+_i) \mid \rho \alpha^+ . \tau^+ \mid \alpha^+
\]

Even in the purely linear language, it is possible to \textit{copy} a value of purely linear type. Define a family of functions

\[
copy_{\tau^+} : \tau^+ \rightarrow (\tau^+ \otimes \tau^+)
\]

such that \(\text{copy}_{\tau^+} v \mapsto (v, v)\) for every \(v : \tau^+\). You do not need to prove this property, just give the definitions of the \textit{copy} functions. Your definitions may be mutually recursive.

We define \(\text{copy}_{\tau^+}\) as:

\[
\begin{align*}
\text{copy}_1 &= \lambda x. (x, \langle \rangle) \\
\text{copy}_{\tau^+_1 \otimes \tau^+_2} &= \lambda p. \text{case } p((p_1, p_2) \Rightarrow \text{case } (\text{copy}_{\tau^+_1} p_1)((x_1, x_2) \Rightarrow \text{case } (\text{copy}_{\tau^+_2} p_2)((y_1, y_2) \Rightarrow \langle \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle \rangle))) \\
\text{copy}_{\oplus_{i \in I}(i : \tau^+_i)} &= \lambda p. \text{case } p(i, p) \Rightarrow \text{case } (\text{copy}_{\tau^+_i} p(p_1, p_2)) \\
\text{copy}_{\rho \alpha^+ . \tau^+} &= \lambda p. \text{case } (\text{copy}_{\rho \alpha . \tau^+} \alpha^+ . \tau^+)(\langle p_1, p_2 \rangle \Rightarrow \langle \langle p_1, p_2 \rangle \rangle)
\end{align*}
\]

There is no value of type \(\alpha^+\), so no need to define \(\text{copy}_{\alpha^+}\). Note that to make this definition terminating, we do not expand the definitions of \(\text{copy}_{\tau^+}\) on the right-hand side; instead we have a call to the functions defined on the left-hand sides.

Task 2 (L22.4, 20 points) A type isomorphism is linear if the functions \textit{Forth} and \textit{Back} are both linear. For each of the following pairs of types provide linear functions witnessing an isomorphism if they exist, or indicate no linear isomorphism exists. You may assume all functions terminate and use either extensional or logical equality as the basis for your judgment.

1. \(\tau \rightarrow (\sigma \rightarrow \rho)\) and \(\sigma \rightarrow (\tau \rightarrow \rho)\)
2. \( \tau \rightarrow (\sigma \rightarrow \rho) \) and \( (\tau \otimes \sigma) \rightarrow \rho \)

\[
\text{Forth} : (\tau \rightarrow (\sigma \rightarrow \rho)) \rightarrow ((\tau \otimes \sigma) \rightarrow \rho)
\]
\[
\text{Forth} = \lambda f. \lambda x. \lambda y. fyx
\]
\[
\text{Back} : ((\tau \otimes \sigma) \rightarrow \rho) \rightarrow (\tau \rightarrow (\sigma \rightarrow \rho))
\]
\[
\text{Back} = \lambda f. \lambda x. \lambda y. fyx
\]

3. \( \tau \rightarrow (\sigma \otimes \rho) \) and \( (\tau \rightarrow \sigma) \otimes (\tau \rightarrow \rho) \)

No linear isomorphism exists.

4. \( (\tau \oplus \sigma) \rightarrow \rho \) and \( (\tau \rightarrow \rho) \otimes (\sigma \rightarrow \rho) \)

No linear isomorphism exists.

5. \( (1 \oplus 1) \rightarrow \tau \) and \( \tau \otimes \tau \)

No linear isomorphism exists.

Linear Processes

Task 3 (L23.2, 20 points) Write a linear function \( \text{inc} \) on the binary representation of natural numbers.

1. Provide the code as a functional expression.
2. Following the conventions of this lecture, show the result of the translation into a process expression. You may use the optimization we presented here. Concretely, define `inc_proc` and `inc` so that the program representation as a configuration would be `!cell inc inc.proc`.

For the sake of simplicity, I ignore `fold` and `unfold` in this Task.

```
incproc = (⟨x, z⟩) ⇒ case x^R (B0 · y ⇒ z^W.B1(y)
  | B1 · y ⇒ d_1 ← (d_2 ← (d_3^W ← inc^R) ;
    d_3 ← (d_3^W ← y^R) ;
    d_2^R.(d_3, d_1))
    z^W.B1(d_1)
  | E · y ⇒ d_1 ← (d_2 ← (d_2^W ← y^R) ;
    d_1^W.E(d_2)) ;
    z^W.B1(d_1))
```

where the initial state of running the program contains `!cell inc inc.proc`

3. Show the initial and final configuration of computation for incrementing the number 1 represented as fold `(B1 · (fold (E · ⟨⟩)))`.

Initial Configuration:

```
!cell inc inc.proc,
cell c_4 ⟨⟩, cell c_3 (E · c_4), cell c_2 (fold c_3),
cell c_1 (B1 · c_2), cell c_0 (fold c_1),
proc d_0 (inc^R.⟨c_0, d_0⟩), cell d_0_`
```

Final Configuration:

```
!cell inc inc.proc,
cell d_0 (fold e_1), cell e_1 (B1 · e_2),
cell e_2 (fold e_3), cell e_3 (B1 · e_4),
cell e_4 (fold e_5), cell e_5 (E · c_4), cell c_4 ⟨⟩
```