Concurrency Semantics
Exercises 6

Prof. Nestmann, 2005

1. Warmup

Output for asynchronous pi Rewrite the labelled transition rule OUT for asynchronous pi processes.

Communication for asynchronous pi Rewrite the reduction rule REACT for asynchronous pi processes.

2. Elastic buffers

We may define an unbounded buffer of names built from cells $\mathsf{E}\langle\, i, o\,\rangle$ as

$$\begin{aligned} \mathsf{E}(\, i, o\,) &:= i(x).\mathsf{F}\langle\, i, o, x\,\rangle \\ \mathsf{F}(\, i, o, x\,) &:= \overline{o}\langle x\rangle.\mathsf{E}\langle\, i, o\,\rangle + i(y).(\boldsymbol{\nu} c)\,(\mathsf{F}\langle\, i, c, y\,\rangle | \mathsf{F}\langle\, c, o, x\,\rangle) \end{aligned}$$

1. Give the transitions for an empty buffer cell performing the following sequence of operations.
   (a) Storing two names
   (b) Reading back one name
   (c) Storing one name
   (d) Reading back one name

   What is the resulting term?

   Note that "useless" bound names can be removed using structural congruence.

2. Using the name-passing capability of the pi-calculus, redefine the cell so that empty cells can disappear when they are next to a full cell.

   Hint: Chose whether disappearance is triggered by having a full cell on the left, on the right or either. You will need another channel.

3. Encoding Polyadic pi

A homomorphic (with respect to parallel composition, restriction and addition) encoding of the polyadic pi calculus into the monadic pi calculus is given by

$$\begin{aligned} [\![\, \overline{a}\langle\vec{b}\rangle.P \,]\!] &\stackrel{\text{def}}{=} \overline{a}\langle b_1\rangle.\cdots.\overline{a}\langle b_n\rangle.[\![\, P \,]\!] \\ [\![\, a(\vec{b}).P \,]\!] &\stackrel{\text{def}}{=} a(b_1).\cdots.a(b_n).[\![\, P \,]\!] \end{aligned}$$

1. Show that for $P \stackrel{\text{def}}{=} (\boldsymbol{\nu} a)\,(\overline{a}\langle a, c\rangle.\overline{a}\langle\rangle.\mathbf{0} \mid a(x, y).x().\mathbf{0})$, $P \rightarrow^* \equiv \mathbf{0}$.
2. Why does the encoding not work for all processes?
   Give an example of a process Q such that Q and $[\![\, Q \,]\!]$ behave differently.
3. Change the definition of $[\![\, \cdot \,]\!]$ such that it does work for all processes P.

   Hint: This couldn't be done in value-passing CCS.

4. Encoding Input-guarded Choice (Homework)

We study the extension of the *asynchronous π-calculus* by *input-guarded* summation

$$P \quad ::= \quad \ldots \quad \Big| \quad \sum_{i \in I} y_i(\vec{x}).P_i \quad \Big| \quad \mathsf{if}\ x\ \mathsf{then}\ P\ \mathsf{else}\ P$$

where, in addition to names, also *primitive boolean values* (denoted $\mathtt{t}$ and $\mathtt{f}$) may be transmitted in communications, and used in the above *conditional operator*.

Find an encoding of this calculus into its summation-free fragment of the calculus, implementing input-guarded summation by means of the other operators. The essential idea is to use parallel composition to run the branches of a summation concurrently, and to use a simple locking mechanism that implements their mutual exclusion, roughly as follows:

$$\begin{aligned} [\![\sum_{i \in I} y_i(\vec{x}).P_i]\!] \quad &\stackrel{\mathrm{def}}{=} \quad (\boldsymbol{\nu} l) \Big(\quad \ldots \quad \Big| \ \prod_{i \in I} [\![y_i(\vec{x}).P_i]\!]_l \Big) \\ [\![y(\vec{x}).P]\!]_l \quad &\stackrel{\mathrm{def}}{=} \end{aligned}$$

where all other operators (including messages) are just translated homomorphically. Note that the translation of an input guard (a branche of a summation) is parameterized on some name l to access the locking mechanism of the summation that it belongs to.

1. Complete the two encoding clauses above.
2. Argue informally for why your solution makes sense.