Pi Calculus

Foundations of Computer Science, Fall 2018

Terms of the π-calculus, ranged over by $P, Q, \ldots$, are written as follows:[1]

$\mathsf{send}\, c\,(\,\vec{e}\,)\, P$	send
$\mathsf{recv}\, c_1\,(\,\vec{x_1}\,)\, P_1 + \cdots + \mathsf{recv}\, c_k\,(\,\vec{x_k}\,)\, P_k$	receive
$(\mathsf{new}\ n)\, P$	private channel creation
$P \mid Q$	parallel composition
$* P$	replication
stop	stop
$N(\,\vec{e}\,)$	process invocation

For the sake of examples, I also add the following terms, though they are not strictly necessary:

$\mathsf{if}\, e\, \mathsf{then}\, P\, \mathsf{else}\, Q$	conditional
$\mathsf{print}(\,e\,)$	printing before stopping

Process definitions take the form

$$N(\,\vec{x}\,) \triangleq P$$

where P is a process that can refer to names in $\vec{x} = x_1, \ldots, x_n$. Process definitions may be recursive. If there are no parameters, we just write $N \triangleq P$.

I assume we have an underlying sequential language for writing expressions e, to express values that can be passed over channels, or checked in conditionals. Generally, it's going to be a simple language of arithmetic expressions and possibly string operations. Note that channel names are also considered values that can be exchanged over channels.

Soda Machine Example: A soda machine that responds to requests to give a Coke or a Pepsi depending on which channel it is communicated with.

[1] This is in fact a restricted version of the π-calculus that only allows choice on receive events. The more general situation that allows choice between arbitrary processes requires a more involved simplification system via labeled transitions.

$$\begin{aligned} \mathit{SodaMachine} \triangleq\ & \mathsf{recv}\ \mathit{getCoke}\,(\ )\ \mathsf{send}\ \mathit{giveOut}\,(\ \texttt{"coke"}\ )\ \mathsf{stop} \\ & + \mathsf{recv}\ \mathit{getPepsi}\,(\ )\ \mathsf{send}\ \mathit{giveOut}\,(\ \texttt{"pepsi"}\ )\ \mathsf{stop} \end{aligned}$$

Some soda customers:

$$\begin{aligned} \mathit{Riccardo} &\triangleq \mathsf{send}\ \mathit{getCoke}\,(\ )\ \mathsf{recv}\ \mathit{giveOut}\,(\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ ) \\ \mathit{Alex} &\triangleq \mathsf{send}\ \mathit{getPepsi}\,(\ )\ \mathsf{recv}\ \mathit{giveOut}\,(\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ ) \end{aligned}$$

Intuitively, when we put together a soda machine and a customer, they should interact, delivering a soda to the customer. That is, we want to determine how the system

$$\mathit{SodaMachine} \mid \mathit{Riccardo}$$

behaves. For that, we need simplification rules.

Simplification Rules: The rules for the core terms are straightforward. Most of the action is handled by the first rule, which says that a send can interact with a receive to exchange a value.

$$\begin{aligned} \mathsf{send}\ c\,(\ \vec{e}\ )\ P \mid \mathsf{recv}\ c_1\,(\ \vec{x_1}\ )\ P_1 + \cdots + \mathsf{recv}\ c_k\,(\ \vec{x_k}\ )\ P_k &= P \mid Q_i\{\vec{e}/\vec{x_i}\} \qquad \text{if } c = c_i \\ P \mid Q &= P' \mid Q \qquad \text{if } P = P' \\ (\mathsf{new}\ n)\ P &= (\mathsf{new}\ n)\ P' \qquad \text{if } P = P' \\ P \mid Q &= Q \mid P \\ (P \mid Q) \mid R &= P \mid (Q \mid R) \\ P \mid \mathsf{stop} &= P \\ (\mathsf{new}\ n)\ \mathsf{stop} &= \mathsf{stop} \\ (\mathsf{new}\ n)\ (\mathsf{new}\ m)\ P &= (\mathsf{new}\ m)\ (\mathsf{new}\ n)\ P \\ (\mathsf{new}\ n)\ (P \mid Q) &= ((\mathsf{new}\ n)\ P) \mid Q \qquad \text{if } n \notin \mathit{fn}(Q) \end{aligned}$$

Function *fn* returns the free names in a process, that is, those names that are not in the scope of a new or a recv.

Because parallel composition is associative, $(P \mid Q) \mid R = P \mid (Q \mid R)$, we can simply write $P \mid Q \mid R$, and more generally, $P_1 \mid \ \ldots\ \mid P_k$ without ambiguity.

Invoking a process simplifies to its definition: if $N(\ \vec{x}\ ) \triangleq P$, then:

$$N(\ \vec{e}\ ) = P\{\vec{e}/\vec{x}\}$$

Our additional terms also have simple simplification rules:

$$\begin{aligned}
\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q &= P && \text{if } e \text{ evaluates to true} \\
\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q &= Q && \text{if } e \text{ does not evaluate to true} \\
\mathsf{print}(\, e\,) &= \mathsf{stop} && \text{(with a side effect of printing } e\text{)}
\end{aligned}$$

The following rules are not technically needed, since they can be derived from the rules above, but they make it easier to simplify examples by hand. Mostly, they are permuted forms of the above rules:

$$\mathsf{recv}\, c_1\, (\, \vec{x_1}\,)\, P_1 + \cdots + \mathsf{recv}\, c_k\, (\, \vec{x_k}\,)\, P_k \mid \mathsf{send}\, c\, (\, \vec{e}\,)\, P = P \mid Q_i\{\vec{e}/\vec{x_i}\} \qquad \text{if } c = c_i$$

$$\begin{aligned}
P \mid Q &= P \mid Q' && \text{if } Q = Q' \\
\mathsf{stop} \mid Q &= Q \\
(\mathsf{new}\ n)\,(P \mid Q) &= P \mid ((\mathsf{new}\ n)\, Q) && \text{if } n \notin \mathit{fn}(P) \\
P_1 \mid \ldots P_k &= P_{\pi(1)} \mid \ldots \mid P_{\pi(k)} && \text{for } \pi \text{ a permutation}
\end{aligned}$$

The last simplification says that you can reorder processes in any big parallel composition. This is useful to bring two processes that need to interact next to each other.

Unlike for the lambda calculus, simplifications are not deterministic. In particular, if we track what gets printed during simplification, we may get different sequences of things being printed on the screen. For example, parallel composition can lead to a simplification of the left subterm or the right subterm first:

$$\mathsf{print}(\,\texttt{"a"}\,) \mid \mathsf{print}(\,\texttt{"b"}\,)$$

will simplify either to:

$$\begin{aligned}
&\mathsf{print}(\,\texttt{"a"}\,) \mid \mathsf{print}(\,\texttt{"b"}\,) && \Rightarrow \texttt{a} \\
&\quad = \mathsf{stop} \mid \mathsf{print}(\,\texttt{"b"}\,) \\
&\quad = \mathsf{print}(\,\texttt{"b"}\,) && \Rightarrow \texttt{b} \\
&\quad = \mathsf{stop}
\end{aligned}$$

or to:

$$\begin{aligned}
&\mathsf{print}(\,\texttt{"a"}\,) \mid \mathsf{print}(\,\texttt{"b"}\,) && \Rightarrow \texttt{b} \\
&\quad = \mathsf{print}(\,\texttt{"a"}\,) \mid \mathsf{stop} \\
&\quad = \mathsf{print}(\,\texttt{"a"}\,) && \Rightarrow \texttt{a} \\
&\quad = \mathsf{stop}
\end{aligned}$$

You can think of these two simplifications as the possible behaviors of the process. Concurrent processes will generally have multiple behaviors. Managing those possible behaviors is the art and science of concurrent programming.

Communication may also involve different possible simplifications, such as the following process:

$$\mathsf{send}\, c\,(\,)\,\mathsf{stop} \mid \mathsf{recv}\, c\,(\,)\,\mathsf{print}(\,\texttt{"a"}\,) + \mathsf{recv}\, c\,(\,)\,\mathsf{print}(\,\texttt{"b"}\,)$$

which can simplify as:

$$\begin{aligned}
&\mathsf{send}\, c\,(\,)\,\mathsf{stop} \mid \mathsf{recv}\, c\,(\,)\,\mathsf{print}(\,\texttt{"a"}\,) + \mathsf{recv}\, c\,(\,)\,\mathsf{print}(\,\texttt{"b"}\,) \\
&\quad = \mathsf{stop} \mid \mathsf{print}(\,\texttt{"a"}\,) \\
&\quad = \mathsf{print}(\,\texttt{"a"}\,) && \Rightarrow \texttt{a} \\
&\quad = \mathsf{stop}
\end{aligned}$$

or as:

$$\begin{aligned}
&\mathsf{send}\, c\,(\,)\,\mathsf{stop} \mid \mathsf{recv}\, c\,(\,)\,\mathsf{print}(\,\texttt{"a"}\,) + \mathsf{recv}\, c\,(\,)\,\mathsf{print}(\,\texttt{"b"}\,) \\
&\quad = \mathsf{stop} \mid \mathsf{print}(\,\texttt{"b"}\,) \\
&\quad = \mathsf{print}(\,\texttt{"b"}\,) && \Rightarrow \texttt{b} \\
&\quad = \mathsf{stop}
\end{aligned}$$

Soda Machine Example Revisited:

$$\begin{aligned}
&\mathit{SodaMachine} \mid \mathit{Riccardo} \\
&\quad = (\mathsf{recv}\, \mathit{getCoke}\,(\,)\,\mathsf{send}\, \mathit{giveOut}\,(\,\texttt{"coke"}\,)\,\mathsf{stop} \\
&\qquad + \mathsf{recv}\, \mathit{getPepsi}\,(\,)\,\mathsf{send}\, \mathit{giveOut}\,(\,\texttt{"pepsi"}\,)\,\mathsf{stop}) \\
&\qquad \mid \mathsf{send}\, \mathit{getCoke}\,(\,)\,\mathsf{recv}\, \mathit{giveOut}\,(\,y\,)\,\mathsf{print}(\,\texttt{"R got"} + y\,) \\
&\quad = \mathsf{send}\, \mathit{giveOut}\,(\,\texttt{"coke"}\,)\,\mathsf{stop} \mid \mathsf{recv}\, \mathit{giveOut}\,(\,y\,)\,\mathsf{print}(\,\texttt{"R got"} + y\,) \\
&\quad = \mathsf{stop} \mid \mathsf{print}(\,\texttt{"R got coke"}\,) \\
&\quad = \mathsf{print}(\,\texttt{"R got coke"}\,) && \Rightarrow \texttt{R got coke} \\
&\quad = \mathsf{stop}
\end{aligned}$$

But we have a problem:

$$\begin{aligned}
&\mathit{SodaMachine} \mid \mathit{Riccardo} \mid \mathit{Alex} \\
&\quad = (\mathsf{recv}\, \mathit{getCoke}\,(\,)\,\mathsf{send}\, \mathit{giveOut}\,(\,\texttt{"coke"}\,)\,\mathsf{stop} \\
&\qquad + \mathsf{recv}\, \mathit{getPepsi}\,(\,)\,\mathsf{send}\, \mathit{giveOut}\,(\,\texttt{"pepsi"}\,)\,\mathsf{stop}) \\
&\qquad \mid \mathsf{send}\, \mathit{getCoke}\,(\,)\,\mathsf{recv}\, \mathit{giveOut}\,(\,y\,)\,\mathsf{print}(\,\texttt{"R got"} + y\,) \mid \mathit{Alex} \\
&\quad = \mathsf{send}\, \mathit{giveOut}\,(\,\texttt{"coke"}\,)\,\mathsf{stop} \mid \mathsf{recv}\, \mathit{giveOut}\,(\,y\,)\,\mathsf{print}(\,\texttt{"R got"} + y\,) \mid \mathit{Alex} \\
&\quad = \mathsf{stop} \mid \mathsf{print}(\,\texttt{"R got coke"}\,) \mid \mathit{Alex} \\
&\quad = \mathsf{print}(\,\texttt{"R got coke"}\,) \mid \mathit{Alex} && \Rightarrow \texttt{R got coke}
\end{aligned}$$

$= \mathsf{stop} \mid \mathit{Alex}$
$= \mathit{Alex}$

Alex is stuck at the end without the possibility of getting a soda. Our soda machine is a one-shot gadget. We can fix that with replication:

$$\mathit{MultiSodaMachine} \triangleq * \mathit{SodaMachine}$$

And here's a sample simplification:

$\mathit{MultiSodaMachine} \mid \mathit{Riccardo} \mid \mathit{Alex}$
$= * \mathit{SodaMachine} \mid \mathit{Riccardo} \mid \mathit{Alex}$
$= \mathit{SodaMachine} \mid * \mathit{SodaMachine} \mid \mathit{Riccardo} \mid \mathit{Alex}$
$= \mathit{SodaMachine} \mid \mathit{Riccardo} \mid * \mathit{SodaMachine} \mid \mathit{Alex}$
$= (\mathsf{recv}\ \mathit{getCoke}\ (\ )\ \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop}$
$\quad + \mathsf{recv}\ \mathit{getPepsi}\ (\ )\ \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop})$
$\quad \mid \mathsf{send}\ \mathit{getCoke}\ (\ )\ \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ )$
$\quad \mid * \mathit{SodaMachine} \mid \mathit{Alex}$
$= \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ )$
$\quad \mid * \mathit{SodaMachine} \mid \mathit{Alex}$
$= \mathsf{stop} \mid \mathsf{print}(\ \texttt{"R got coke"}\ ) \mid * \mathit{SodaMachine} \mid \mathit{Alex}$
$= \mathsf{print}(\ \texttt{"R got coke"}\ ) \mid * \mathit{SodaMachine} \mid \mathit{Alex} \qquad \Rightarrow \texttt{R got coke}$
$= \mathsf{stop} \mid * \mathit{SodaMachine} \mid \mathit{Alex}$
$= * \mathit{SodaMachine} \mid \mathit{Alex}$
$= \mathit{SodaMachine} \mid * \mathit{SodaMachine} \mid \mathit{Alex}$
$= \mathit{SodaMachine} \mid \mathit{Alex} \mid * \mathit{SodaMachine}$
$= (\mathsf{recv}\ \mathit{getCoke}\ (\ )\ \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop}$
$\quad + \mathsf{recv}\ \mathit{getPepsi}\ (\ )\ \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop})$
$\quad \mid \mathsf{send}\ \mathit{getPepsi}\ (\ )\ \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ )$
$\quad \mid * \mathit{SodaMachine}$
$= \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop} \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ )$
$\quad \mid * \mathit{SodaMachine}$
$= \mathsf{stop} \mid \mathsf{print}(\ \texttt{"A got pepsi"}\ ) \mid * \mathit{SodaMachine}$
$= \mathsf{print}(\ \texttt{"A got pepsi"}\ ) \mid * \mathit{SodaMachine} \qquad \Rightarrow \texttt{A got pepsi}$
$= \mathsf{stop}\mathit{Par} * \mathit{SodaMachine}$
$= * \mathit{SodaMachine}$

But we still have a problem. Here is another sequence of simplifications that exposes it:

$$
\begin{aligned}
&\mathit{MultiSodaMachine} \mid \mathit{Riccardo} \mid \mathit{Alex} \\
&\quad = * \mathit{SodaMachine} \mid \mathit{Riccardo} \mid \mathit{Alex} \\
&\quad = \mathit{SodaMachine} \mid * \mathit{SodaMachine} \mid \mathit{Riccardo} \mid \mathit{Alex} \\
&\quad = \mathit{SodaMachine} \mid \mathit{Riccardo} \mid * \mathit{SodaMachine} \mid \mathit{Alex} \\
&\quad = (\mathsf{recv}\ \mathit{getCoke}\ (\ )\ \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\quad + \mathsf{recv}\ \mathit{getPepsi}\ (\ )\ \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop}) \\
&\qquad \mid \mathsf{send}\ \mathit{getCoke}\ (\ )\ \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ ) \\
&\qquad \mid * \mathit{SodaMachine} \mid \mathit{Alex} \\
&\quad = \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ ) \\
&\qquad \mid * \mathit{SodaMachine} \mid \mathit{Alex} \\
&\quad = \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ ) \\
&\qquad \mid \mathit{SodaMachine} \mid * \mathit{SodaMachine} \mid \mathit{Alex} \\
&\quad = \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ ) \\
&\qquad \mid \mathit{SodaMachine} \mid \mathit{Alex} \mid * \mathit{SodaMachine} \\
&\quad = \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ ) \\
&\qquad \mid (\mathsf{recv}\ \mathit{getCoke}\ (\ )\ \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\quad + \mathsf{recv}\ \mathit{getPepsi}\ (\ )\ \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop}) \\
&\qquad \mid \mathsf{send}\ \mathit{getPepsi}\ (\ )\ \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ ) \\
&\qquad \mid * \mathit{SodaMachine} \\
&\quad = \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ ) \\
&\qquad \mid \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop} \\
&\qquad \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ ) \\
&\qquad \mid * \mathit{SodaMachine} \\
&\quad = \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad \mid \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop} \\
&\qquad \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ ) \\
&\qquad \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ ) \\
&\qquad \mid * \mathit{SodaMachine}
\end{aligned}
$$

$$
\begin{aligned}
&= \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\quad \mid \mathsf{stop} \\
&\quad \mid \mathsf{print}(\ \texttt{"R got pepsi"}\ ) \\
&\quad \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ ) \\
&\quad \mid *\ \mathit{SodaMachine} \\
&= \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} && \Rightarrow \texttt{R got pepsi} \\
&\quad \mid \mathsf{print}(\ \texttt{"R got pepsi"}\ ) \\
&\quad \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ ) \\
&\quad \mid *\ \mathit{SodaMachine} \\
&= \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\quad \mid \mathsf{stop} \\
&\quad \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ ) \\
&\quad \mid *\ \mathit{SodaMachine} \\
&= \mathsf{send}\ \mathit{giveOut}\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\quad \mid \mathsf{recv}\ \mathit{giveOut}\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ ) \\
&\quad \mid *\ \mathit{SodaMachine} \\
&= \mathsf{stop} \mid \mathsf{print}(\ \texttt{"A got coke"}\ ) \mid *\ \mathit{SodaMachine} \\
&= \mathsf{print}(\ \texttt{"A got coke"}\ ) \mid *\ \mathit{SodaMachine} && \Rightarrow \texttt{A got coke} \\
&= \mathsf{stop} \mid *\ \mathit{SodaMachine} \\
&= \mid *\ \mathit{SodaMachine}
\end{aligned}
$$

This is not good: Riccardo asked for a Coke and got a Pepsi, while Alex asked for a Pepsi and got a Coke. The cause is obvious: response channel *giveOut* from the soda machine is a global resource that anybody can read from. In fact, we can easily write a process that would "steal" sodas, that could intercept them before they get delivered to their intended recipient.

The fix is to change the "protocol" with which we interact with the soda machine. Rather than relying on a global response channel, a customer interested in a soda should send along with their request a channel which the soda machine will use to deliver the soda. Creating that reply channel privately and not sharing it ensures that no other process can intercept the purchased soda.

$$
\begin{aligned}
&\mathit{MultiSodaMachine} \triangleq *\ \mathit{SodaMachine} \\
&\mathit{SodaMachine} \triangleq\ \mathsf{recv}\ \mathit{getCoke}\ (\ r\ )\ \mathsf{send}\ r\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\qquad + \mathsf{recv}\ \mathit{getPepsi}\ (\ r\ )\ \mathsf{send}\ r\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop}
\end{aligned}
$$

A customer now needs to create a reply channel before requesting a soda:

$$Riccardo \triangleq (\mathsf{new}\ c)\, \mathsf{send}\ getCoke\ (\, c\,)\ \mathsf{recv}\ c\ (\, y\,)\ \mathsf{print}(\ \texttt{"R got"} + y\,)$$
$$Alex \triangleq (\mathsf{new}\ d)\, \mathsf{send}\ getPepsi\ (\, d\,)\ \mathsf{recv}\ d\ (\, y\,)\ \mathsf{print}(\ \texttt{"A got"} + y\,)$$

(I'm using two different names for the private channels created in each process, in order to clarify our sample simplifications below. But they could use the same name, of course, since they are private. Run through the simplifications in that case if you'd like to see what happens.)

Simple interactions proceed as straightforwardly as before. Watch where we move the $(\mathsf{new}\ c)$ around to allow for the interaction of the send and receive on channel *getCoke*:

$$\begin{aligned}
&MultiSodaMachine \mid Riccardo \\
&\quad = *\, SodaMachine \mid Riccardo \\
&\quad = SodaMachine \mid *\, SodaMachine \mid Riccardo \\
&\quad = SodaMachine \mid Riccardo \mid *\, SodaMachine \\
&\quad = (\mathsf{recv}\ getCoke\ (\, r\,)\ \mathsf{send}\ r\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\quad + \mathsf{recv}\ getPepsi\ (\, r\,)\ \mathsf{send}\ r\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop}) \\
&\qquad \mid (\mathsf{new}\ c)\, \mathsf{send}\ getCoke\ (\, c\,)\ \mathsf{recv}\ c\ (\, y\,)\ \mathsf{print}(\ \texttt{"R got"} + y\,) \\
&\qquad \mid *\, SodaMachine \\
&\quad = (\mathsf{new}\ c)\ (\mathsf{recv}\ getCoke\ (\, r\,)\ \mathsf{send}\ r\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\qquad\quad + \mathsf{recv}\ getPepsi\ (\, r\,)\ \mathsf{send}\ r\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop}) \\
&\qquad\qquad \mid \mathsf{send}\ getCoke\ (\, c\,)\ \mathsf{recv}\ c\ (\, y\,)\ \mathsf{print}(\ \texttt{"R got"} + y\,)) \\
&\qquad \mid *\, SodaMachine \\
&\quad = (\mathsf{new}\ c)\, (\mathsf{send}\ c\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \mid \mathsf{recv}\ c\ (\, y\,)\ \mathsf{print}(\ \texttt{"R got"} + y\,)) \\
&\qquad \mid *\, SodaMachine \\
&\quad = (\mathsf{new}\ c)\, (\mathsf{stop} \mid \mathsf{print}(\ \texttt{"R got coke"}\ )) \mid *\, SodaMachine \\
&\quad = (\mathsf{new}\ c)\, \mathsf{print}(\ \texttt{"R got coke"}\ ) \mid *\, SodaMachine \qquad\qquad \Rightarrow \texttt{R got coke} \\
&\quad = (\mathsf{new}\ c)\, \mathsf{stop} \mid *\, SodaMachine \\
&\quad = \mathsf{stop} \mid *\, SodaMachine \\
&\quad = *\, SodaMachine
\end{aligned}$$

More interestingly, the problem we identified earlier about two customers asking for a soda each and getting the wrong soda delivered cannot occur, as there is no choice of which communications to performs at any point, as the following example shows.

$$MultiSodaMachine \mid Riccardo \mid Alex$$

$$
\begin{aligned}
&= *\,SodaMachine \mid Riccardo \mid Alex \\
&= SodaMachine \mid *\,SodaMachine \mid Riccardo \mid Alex \\
&= SodaMachine \mid Riccardo \mid *\,SodaMachine \mid Alex \\
&= (\mathsf{recv}\ getCoke\ (\ r\ )\ \mathsf{send}\ r\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad + \mathsf{recv}\ getPepsi\ (\ r\ )\ \mathsf{send}\ r\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop}) \\
&\quad \mid (\mathsf{new}\ c)\ \mathsf{send}\ getCoke\ (\ c\ )\ \mathsf{recv}\ c\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ ) \\
&\quad \mid *\,SodaMachine \mid Alex \\
&= (\mathsf{new}\ c)\ (\mathsf{recv}\ getCoke\ (\ r\ )\ \mathsf{send}\ r\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\qquad + \mathsf{recv}\ getPepsi\ (\ r\ )\ \mathsf{send}\ r\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop}) \\
&\qquad\quad \mid \mathsf{send}\ getCoke\ (\ c\ )\ \mathsf{recv}\ c\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ )) \\
&\quad \mid *\,SodaMachine \mid Alex \\
&= (\mathsf{new}\ c)\ (\mathsf{send}\ c\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\quad \mid \mathsf{recv}\ c\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ )) \\
&\quad \mid *\,SodaMachine \mid Alex \\
&= (\mathsf{new}\ c)\ (\mathsf{send}\ c\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\quad \mid \mathsf{recv}\ c\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ )) \\
&\quad \mid SodaMachine \mid *\,SodaMachine \mid Alex \\
&= (\mathsf{new}\ c)\ (\mathsf{send}\ c\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\quad \mid \mathsf{recv}\ c\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ )) \\
&\quad \mid SodaMachine \mid Alex \mid *\,SodaMachine \\
&= (\mathsf{new}\ c)\ (\mathsf{send}\ c\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\quad \mid \mathsf{recv}\ c\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ )) \\
&\quad \mid (\mathsf{recv}\ getCoke\ (\ r\ )\ \mathsf{send}\ r\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad + \mathsf{recv}\ getPepsi\ (\ r\ )\ \mathsf{send}\ r\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop}) \\
&\quad \mid (\mathsf{new}\ d)\ \mathsf{send}\ getPepsi\ (\ d\ )\ \mathsf{recv}\ d\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ ) \\
&\quad \mid *\,SodaMachine \\
&= (\mathsf{new}\ c)\ (\mathsf{send}\ c\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\quad \mid \mathsf{recv}\ c\ (\ y\ )\ \mathsf{print}(\ \texttt{"R got"} + y\ )) \\
&\quad \mid (\mathsf{new}\ d)\ (\mathsf{recv}\ getCoke\ (\ r\ )\ \mathsf{send}\ r\ (\ \texttt{"coke"}\ )\ \mathsf{stop} \\
&\qquad\qquad + \mathsf{recv}\ getPepsi\ (\ r\ )\ \mathsf{send}\ r\ (\ \texttt{"pepsi"}\ )\ \mathsf{stop}) \\
&\qquad\quad \mid \mathsf{send}\ getPepsi\ (\ d\ )\ \mathsf{recv}\ d\ (\ y\ )\ \mathsf{print}(\ \texttt{"A got"} + y\ )) \\
&\quad \mid *\,SodaMachine
\end{aligned}
$$

$$
\begin{aligned}
&= (\mathsf{new}\ c)\ (\mathsf{send}\, c\, (\ \texttt{"coke"}\ )\, \mathsf{stop} \\
&\qquad\qquad \mid \mathsf{recv}\, c\, (\, y\,)\, \mathsf{print}(\ \texttt{"R got"} + y\,)) \\
&\quad \mid (\mathsf{new}\ d)\, (\, \mathsf{send}\, d\, (\ \texttt{"pepsi"}\ )\, \mathsf{stop} \\
&\qquad\qquad \mid \mathsf{recv}\, d\, (\, y\,)\, \mathsf{print}(\ \texttt{"A got"} + y\,)) \\
&\quad \mid *\ \mathit{SodaMachine} \\
&= (\mathsf{new}\ c)\, (\mathsf{stop} \mid \mathsf{print}(\ \texttt{"R got coke"}\ )) \\
&\quad \mid (\mathsf{new}\ d)\ (\mathsf{send}\, d\, (\ \texttt{"pepsi"}\ )\, \mathsf{stop} \\
&\qquad\qquad \mid \mathsf{recv}\, d\, (\, y\,)\, \mathsf{print}(\ \texttt{"A got"} + y\,)) \\
&\quad \mid *\ \mathit{SodaMachine} \\
&= (\mathsf{new}\ c)\, \mathsf{print}(\ \texttt{"R got coke"}\ ) && \Rightarrow \texttt{R got coke} \\
&\quad \mid (\mathsf{new}\ d)\ (\mathsf{send}\, d\, (\ \texttt{"pepsi"}\ )\, \mathsf{stop} \\
&\qquad\qquad \mid \mathsf{recv}\, d\, (\, y\,)\, \mathsf{print}(\ \texttt{"A got"} + y\,)) \\
&\quad \mid *\ \mathit{SodaMachine} \\
&= (\mathsf{new}\ c)\, \mathsf{stop} \\
&\quad \mid (\mathsf{new}\ d)\ (\mathsf{send}\, d\, (\ \texttt{"pepsi"}\ )\, \mathsf{stop} \\
&\qquad\qquad \mid \mathsf{recv}\, d\, (\, y\,)\, \mathsf{print}(\ \texttt{"A got"} + y\,)) \\
&\quad \mid *\ \mathit{SodaMachine} \\
&= \mathsf{stop} \\
&\quad \mid (\mathsf{new}\ d)\ (\mathsf{send}\, d\, (\ \texttt{"pepsi"}\ )\, \mathsf{stop} \\
&\qquad\qquad \mid \mathsf{recv}\, d\, (\, y\,)\, \mathsf{print}(\ \texttt{"A got"} + y\,)) \\
&\quad \mid *\ \mathit{SodaMachine} \\
&= (\mathsf{new}\ d)\ (\mathsf{send}\, d\, (\ \texttt{"pepsi"}\ )\, \mathsf{stop} \\
&\qquad\qquad \mid \mathsf{recv}\, d\, (\, y\,)\, \mathsf{print}(\ \texttt{"A got"} + y\,)) \\
&\quad \mid *\ \mathit{SodaMachine} \\
&= (\mathsf{new}\ d)\, (\mathsf{stop} \mid \mathsf{print}(\ \texttt{"A got pepsi"}\ )) \mid *\ \mathit{SodaMachine} \\
&= (\mathsf{new}\ d)\, \mathsf{print}(\ \texttt{"A got pepsi"}\ ) \mid *\ \mathit{SodaMachine} && \Rightarrow \texttt{A got pepsi} \\
&= (\mathsf{new}\ d)\, \mathsf{stop} \mid *\ \mathit{SodaMachine} \\
&= \mathsf{stop} \mid *\ \mathit{SodaMachine} \\
&= *\, \mathit{SodaMachine}
\end{aligned}
$$

Authentication Server Example: An authentication server is a service that provides authentication to other serices: a client can authenticate itself once using a name and password, and gets a token that it can pass to other service. Those service contact the authentication server to verify the token, and if it checks out, can grant access to the services they offer. Only the authentication server needs to store the password of the client.

To model such a structure, we represent the authentication server as a process that expects a name and a password on a global *login* channel. After verifying that the name and password are correct — we abstract this away with a predicate $ok()$ — the authentication server creates a token (which is just a channel) and returns that token to the client. The token channel can be used by a service to contact the authentication server to confirm the identity of a client.

$$
\begin{aligned}
\mathit{Auth} \triangleq\ & * \ \mathsf{recv}\ \mathit{login}\ (\ \mathit{name}, \mathit{pwd}, r\ ) \\
& \quad \mathsf{if}\ \mathit{ok}(\mathit{name}, \mathit{pwd}) \\
& \quad\quad \mathsf{then}\ (\mathsf{new}\ \mathit{token})\ (\mathsf{send}\ r\ (\ \mathit{token}\ )\ \mathsf{stop} \\
& \quad\quad\quad\quad\quad\quad \mid\ * \ \mathsf{recv}\ \mathit{token}\ (\ n, r\ ) \\
& \quad\quad\quad\quad\quad\quad\quad\quad \mathsf{if}\ n = \mathit{name} \\
& \quad\quad\quad\quad\quad\quad\quad\quad\quad \mathsf{then}\ \mathsf{send}\ r\ (\ \texttt{"ok"}\ )\ \mathsf{stop} \\
& \quad\quad\quad\quad\quad\quad\quad\quad \mathsf{else}\ \mathsf{send}\ r\ (\ \texttt{"fail"}\ )\ \mathsf{stop}) \\
& \quad \mathsf{else}\ \mathsf{stop} \\
\mathit{Service} \triangleq\ & * \ \mathsf{recv}\ \mathit{access}\ (\ \mathit{name}, \mathit{token}\ ) \\
& \quad\quad (\mathsf{new}\ r)\ \mathsf{send}\ \mathit{token}\ (\ \mathit{name}, r\ ) \\
& \quad\quad\quad\quad\quad\quad \mathsf{recv}\ r\ (\ y\ ) \\
& \quad\quad\quad\quad\quad\quad \mathsf{if}\ y = \text{"}\mathit{ok}\text{"} \\
& \quad\quad\quad\quad\quad\quad\quad\quad \mathsf{then}\ \mathsf{print}(\ \texttt{"Render service to"} + \mathit{name}\ ) \\
& \quad\quad\quad\quad\quad\quad \mathsf{else}\ \mathsf{print}(\ \texttt{"Unauthorized access"}\ ) \\
\mathit{Dora} \triangleq\ & (\mathsf{new}\ r)\ \mathsf{send}\ \mathit{login}\ (\ \texttt{"dora"}, \texttt{"1234"}, r\ )\ \mathsf{recv}\ r\ (\ \mathit{token}\ )\ \mathsf{send}\ \mathit{access}\ (\ \texttt{"dora"}, \mathit{token}\ ) \\
\mathit{Swiper} \triangleq\ & (\mathsf{new}\ r)\ \mathsf{send}\ \mathit{login}\ (\ \texttt{"swiper"}, \texttt{"9999"}, r\ )\ \mathsf{recv}\ r\ (\ \mathit{token}\ )\ \mathsf{send}\ \mathit{access}\ (\ \texttt{"dora"}, \mathit{token}\ )
\end{aligned}
$$

Running through the simplifications shows that

$$\mathit{Auth} \mid \mathit{Service} \mid \mathit{Dora} \mid \mathit{Swiper}$$

correctly prints $\Rightarrow$ `Render service to dora` and $\Rightarrow$ `Unauthorized access` in some order before becoming

$$\mathit{Auth} \mid \mathit{Service}$$

indicating that *Dora* could access the service, while *Swiper* could not.