Exercises - 1

Forward-Only CCS

The interest and history of the Calculus of Communicating Systems (CCS) is given e.g. at https://en.wikipedia.org/wiki/Calculus_of_communicating_systems.

Syntax

(Co-)names and labels Let $\mathsf{N} = \{a, b, c, \dots\}$ be a set of *names* and $\overline{\mathsf{N}} = \{\overline{a}, \overline{b}, \overline{c}, \dots\}$ its set of *co-names*. The set of *channels* C is $\mathsf{N} \cup \overline{\mathsf{N}}$, the set of *labels* L is $\mathsf{N} \cup \overline{\mathsf{N}} \cup \{\tau\}$, and we use α, β (resp. λ) to range over L (resp. C). A bijection $\overline{\cdot} : \mathsf{C} \to \overline{\mathsf{C}}$, whose inverse is also written $\overline{\cdot}$, gives the *complement* of a channel.

The intuition is that a channel represents a port, a slot, a button, a switch, an action…. A name represents the action of *sending an output* along that channel, and a co-name represents the action of *receiving an input* along that channel (or the other way around: it does not change much since $\overline{\cdot}$ is an involution). Names and co-names are *complement* because receiving an input on $\overline{a}$ suppose that a message was output on a. When such a *synchronization* happen, the symbol τ is used to represent a silent action, something that cannot be seen by the rest of the world.

CCS is not interested with the content of the messages that are exchanged: they are treated as black boxes. It only cares about the possibility of *processes* (read: computers, threads, agents, …) interacting.

Processes are then constructed in the following way:

$$P, Q ::= 0 \parallel \alpha.P \parallel P + Q \parallel P|Q \parallel P[\alpha/\beta] \parallel P\backslash\alpha \parallel A$$

meaning that

- 0 (the inactive process) is a process,
- $\alpha.P$ is the process that can send (or receive) along α and then continue as P,
- $P + Q$ is a process that can act as P (exclusive) or as Q,
- $P|Q$ is the process that results from setting in parallel P and Q,
- $P[\alpha/\beta]$ is the process that results from replacing every occurrence of α by β in P,
- $P\backslash\lambda$ is the process P where the channel λ is kept private (that is, it can use it only to exchange messages internally),
- $A \overset{\text{def}}{=} P$ uses the identifier A to refer to the process P (which may contain the identifier A).

This recursive definition, along with replication and recursion, is one of the mechanism in CCS used to represent infinite behaviours.

Exemples: Here are some high-level examples:

- A process $\overline{a}.b.0$ could represent a system that expect a message on port a, send a message on port b, and then terminate. If the message is transferred without being changed, this process represents a forwarder.
- A process $(\overline{a}.a.0)|b.0$ could represent a server that expects a ssh connexion on port a (if the connexion is successful, it would send a message on port a) and *in parallel* wait for printing instructions on port b. This means that the server can do both actions *at the same time*, and in any order: it could first receive the ssh connexion, then receive printing instructions, and finally send the success message.
- A process $\overline{a}.0 + \overline{b}.0$ could represent access to a shared document: a user could log-in on a to edit the document only provided nobody logged-in on b, and reciprocally. The process can accept a connexion on a or on b, but cannot accept both.
- A process $((\overline{a}.0|a.P)|a.Q)\backslash a$ represent a situation where either $a.P$ or $a.Q$ could send a message to $\overline{a}$ and synchronize with it, but nobody else could, as the channel name a is restricted.

- A process $A \overset{\text{def}}{=} \overline{a}.b.A$ is an infinite forwarder: it receives a message on a, send it back on b, and then wait for a message on a again.

Exercise: Usually, we simplify the notation by assuming some convention [1]. We do not write "trailing 0", so that $a.0$ is the same as a, and:

> We assume that the operators have decreasing binding power, in the following order: $\backslash\alpha, \alpha., |, +$.

Explain the meaning of this (slightly modified) quote, and write down the parenthesised version of a couple of terms without parenthesises. For instance, is $a + b|c$ the same as $(a + b)|c$, or the same as $a + (b|c)$? Is $A + a.a|b + c$ the same as $A + ((a.a)|(b + c))$, or is it something else entirely?

Semantics

The processes are then given a *semantics* (a way of *reducing*, of being executed) thanks to a labeled transition system (LTS), given in Figure 1.

Action and Restriction

$$\frac{}{\alpha.P \xrightarrow{\alpha} P}\ \text{act.} \qquad\qquad a \notin \{\alpha, \overline{\alpha}\}\ \frac{P \xrightarrow{\alpha} P'}{P\backslash a \xrightarrow{\alpha} P'\backslash a}\ \text{res.}$$

Parallel Group

$$\frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ |_{\text{L}} \qquad \frac{P \xrightarrow{\lambda} P' \qquad Q \xrightarrow{\overline{\lambda}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}\ \text{syn.} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}\ |_{\text{R}}$$

Sum Group

$$\frac{P \xrightarrow{\alpha} P'}{Q + P \xrightarrow{\alpha} P'}\ +_{\text{L}} \qquad\qquad \frac{P \xrightarrow{\alpha} P'}{Q + P \xrightarrow{\alpha} P'}\ +_{\text{R}}$$

Infinite Group

$$\frac{P \xrightarrow{\alpha} P' \qquad D \overset{\text{def}}{=} P}{D \xrightarrow{\alpha} P'}\ \text{const.}$$

Figure 1: LTS for CCS

This means that to decide, for instance, if $a.P + Q\backslash b$ can become P, we have to find a *derivation* using those rules. In this case, we can (almost!) construct it:

- $a.P$ can use the act. rule to become P,
- as a consequence, $a.P + Q$ can use the $+_{\text{L}}$ rule to become P,
- as a consequence, and since $b \notin \{a, \overline{a}\}$, $a.P + Q\backslash b$ (which is the same as $(a.P + Q)\backslash b$) can use the res. rule to become … $P\backslash b$.

Summarizing, $a.P + Q\backslash b \xrightarrow{a} P\backslash b$, which means that $a.P + Q\backslash b$ can become P once it received a message on a. Note that if we had $a = b$, then the process $a.P$ would be stuck: to make progress, it would have te receive a message on a "from the outside world", but this is not possible because of the restriction: this is what the side condition in the res. rule guarantee.

Exercise: In forward-only CCS, list all the different reductions that $c.(\overline{a}|(b.a|d))$ can perform to reach $0|(0|0)$.

Exercise: (Inspired from [2]) Assuming that $a \neq b$, write the derivation of the transitions:

$$\begin{aligned}(a.P + b.0)|\overline{a}.Q &\to^{a} P|\overline{a}.Q\\ (a.P + b.0)|\overline{a}.Q &\to^{\overline{a}} (a.P + b.0)|Q\\ (a.P + b.0)|a.Q &\to^{b} 0|a.Q\\ ((a.P + b.0)|a.Q)\backslash a &\to^{b} (0|a.Q)\backslash a\\ (a.P|Q)\backslash b|(\overline{a}.Q'\backslash c) &\to^{\tau} (P|Q)\backslash b|(Q'\backslash c)\end{aligned}$$

Vending Machine

In Figure 2 is presented the "canonical example" from [2] (note that the V at the right of the equation are here to trigger recursion).

Exercises: Solve the two exercises, and then consider the following vending machine:

$$V_{\text{bad}} \stackrel{\text{def}}{=} \text{2p.(big.collect.}V_{\text{bad}} + \text{little.collect.little.collect.}V_{\text{bad}})$$

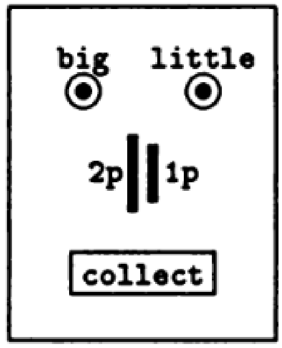
Most people would consider this vending machine as bad: can you explain why?

References

[1] P. Degano, F. Gadducci, C. Priami, Causality and replication in concurrent processes, in: M. Broy, A.V. Zamulin (Eds.), Perspectives of Systems Informatics, 5th International Andrei Ershov Memorial Conference, PSI 2003, Akademgorodok, Novosibirsk, Russia, July 9-12, 2003, Revised Papers, Springer, 2003: pp. 307–318. https://doi.org/10.1007/978-3-540-39866-0_30.

[2] R. Milner, Communication and concurrency, Prentice-Hall, 1989.

Now let us consider a different kind of system: a vending machine. Here (with thanks to Tony Hoare) is a picture of a machine for selling chocolates:



We suppose that a big chocolate costs 2p, a little one costs 1p, and only these coins can be used. One natural way to define the vending machine, V, is in terms of its interaction with the environment at its five ports (2p, `1p`, `big`, `little` and `collect`), as follows:

$$V \stackrel{\text{def}}{=} \texttt{2p.big.collect}.V + \texttt{1p.little.collect}.V$$

This means, for example, that to buy a big chocolate you must put in a 2p coin, press the button marked 'big', and collect your chocolate from the tray. Note already some interesting points:

- There are no parameters involved in any of these actions.
- The machine's behaviour is quite restrictive; it will not let you pay for a big chocolate with two 1p coins, or put in more money before you've collected your purchase.

Exercise 4 Modify V so after that inserting 1p you can either buy a little chocolate or insert 1p more and buy a big one. ■

Exercise 5 Further modify V so that after inserting 2p you can buy either one big chocolate or two little ones. ■

Figure 2: The vending machine example