

Exercises - 1

(Forward-Only) Calculus of Communicating Systems (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 $N = \{a, b, c, \dots\}$ be a set of *names* and $\bar{N} = \{\bar{a}, \bar{b}, \bar{c}, \dots\}$ its set of *co-names*. The set of *channels* C is $N \cup \bar{N}$, the set of *labels* L is $N \cup \bar{N} \cup \{\tau\}$, and we use α, β (resp. λ) to range over L (resp. C). A bijection $\tau : C \rightarrow \bar{C}$, whose inverse is also written τ , 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 τ is an involution). Names and co-names are *complement* because receiving an input on \bar{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:

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

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 ,
- $P \setminus \lambda$ is the process P where the channel λ is kept private (that is, it can use it only to exchange messages internally),
- $A \stackrel{\text{def}}{=} P$ 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.

Examples: Here are some high-level examples:

- A process $\bar{a}.b.0$ 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 $(\bar{a}.a.0)|b.0$ 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 $\bar{a}.0 + \bar{b}.0$ 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 $((\bar{a}.0|a.P)|a.Q) \setminus a$ represent a situation where either $a.P$ or $a.Q$ could send a message to \bar{a} and synchronize with it, but nobody else could, as the channel name a is restricted.

- A process $A \stackrel{\text{def}}{=} \bar{a}.b.A$ could represent an infinite forwarder: it receives a message on a , send it back on b , and then wait for a message on a to forward on b again and again.

Exercise: Write a high-level explanation of the situation represented by the process

$$((a.P_1.\bar{b}.P_2|b.Q_1.Q_2)\backslash b) + ((a.Q_1.\bar{c}.Q_2|c.P_1.P_2)\backslash c)$$

taking inspiration from the following questions:

- Can $b.Q_1.Q_2$ receive a message on b from any other process than $a.P_1.\bar{b}.P_2$?
- Can P_1 and Q_1 execute at the same time?
- Can P_2 and Q_1 execute at the same time?
- Can Q_1 execute twice?

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: $\backslash, \alpha, |, +$.

Explain the meaning of this (slightly modified) quote, and write down the parenthesised version of a couple of terms without parentheses. 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	
$\frac{}{\alpha.P \xrightarrow{\alpha} P} \text{ act.}$	$a \notin \{\alpha, \bar{\alpha}\} \quad \frac{P \xrightarrow{\alpha} P'}{P \backslash a \xrightarrow{\alpha} P' \backslash a} \text{ res.}$
Parallel Group	
$\frac{P \xrightarrow{\alpha} P'}{P Q \xrightarrow{\alpha} P' Q} _L$	$\frac{P \xrightarrow{\lambda} P' \quad Q \xrightarrow{\bar{\lambda}} Q'}{P Q \xrightarrow{\tau} P' Q'} \text{ syn.} \quad \frac{Q \xrightarrow{\alpha} Q'}{P Q \xrightarrow{\alpha} P Q'} _R$
Sum Group	
$\frac{P \xrightarrow{\alpha} P'}{Q + P \xrightarrow{\alpha} P'} +_L$	$\frac{P \xrightarrow{\alpha} P'}{Q + P \xrightarrow{\alpha} P'} +_R$
Infinite Group	
$\frac{P \xrightarrow{\alpha} P' \quad D \stackrel{\text{def}}{=} P}{D \xrightarrow{\alpha} P'} \text{ const.}$	

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 $+_L$ rule to become P ,
- as a consequence, and since $b \notin \{a, \bar{a}\}$, $a.P + Q \setminus b$ (which is the same as $(a.P + Q) \setminus b$) can use the res. rule to become ... $P \setminus b$.

Summarizing, $a.P + Q \setminus b \xrightarrow{a} P \setminus b$, which means that $a.P + Q \setminus b$ can become $P \setminus b$ 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 to 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.

Examples: Note that a process $a|\bar{a}$ can either

- Perform a , then \bar{a} ,
- Perform \bar{a} , then a ,
- Perform τ , which corresponds to a synchronization.

In short, it means that the process $a|\bar{a}$ can either communicate with the outside world, or “keep it between then”. Note that the process $(a|\bar{a}) \setminus a$ can only perform the τ transition.

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

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

$$\begin{aligned}
& (a.P + b.0)|\bar{a}.Q \xrightarrow{a} P|\bar{a}.Q \\
& (a.P + b.0)|\bar{a}.Q \xrightarrow{\bar{a}} (a.P + b.0)|Q \\
& (a.P + b.0)|a.Q \xrightarrow{b} 0|a.Q \\
& ((a.P + b.0)|a.Q) \setminus a \xrightarrow{b} (0|a.Q) \setminus a \\
& (a.P|Q) \setminus b | (\bar{a}.Q' \setminus c) \xrightarrow{\tau} (P|Q) \setminus b | (Q' \setminus c)
\end{aligned}$$

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:

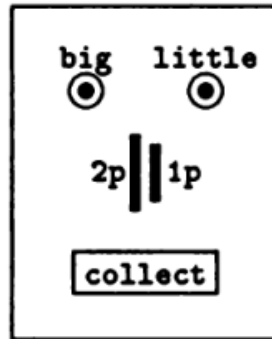
$$V_{\text{bad}} \stackrel{\text{def}}{=} 2p.(\text{big.collect}.V_{\text{bad}} + \text{little.collect.little.collect}.V_{\text{bad}})$$

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:

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

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