Université Paris Cité
Master 2 : Logique Mathématique et Fondements de l'Informatique

Coq can communicate

By Gaëtan Lopez

Internship's thesis of Computer Science and Mathematics

Directed by Giovanni BERNARDI
And by Hugo FEREE
At the Research Institute on the Foundations of Computer Science (IRIF)
Presented and defended on 16/09/2024

Contents

Chapter 0

Introduction

MUST-preorder Code refactoring is a routine task, necessary to either update or develop software. Correct refactoring in turn requires ensuring that a (new) program q can be used in place of a program p. Usually this is tackled via refinement relations. In the setting of deterministic programming languages, the most studied refinement is the preorder defined by Morris [29] (pag.50), by letting $p \leqslant q$ if for all contexts C, whenever $C(\!|p|\!)$ evaluates to a value v, then $C(\!|q|\!)$ also evaluates to v. In the setting of nondeterministic client-server systems it is natural to reformulate this preorder by replacing evaluation to a value with a suitable liveness property [30] [31]. Let $p \parallel r$ denote a parallel composition in which the identities of the server p and the client r are distinguished, and whose computations have the form $p \parallel r \longrightarrow p_1 \parallel r_1 \longrightarrow p_2 \parallel r_2 \longrightarrow ...$ where each step represents either an internal computation of one of the two components, or an *interaction* between them. We express liveness by saying that p *must pass* r if in every maximal execution of $p \parallel r$, there exists a state $p_i \parallel r_i$ such that GOOD(r_i), where GOOD is a decidable predicate indicating that the client has reached a successful state. Servers are then compared according to their capacity to satisfy clients, i.e. to lead them to successful states. This is formalised by the MUST-preorder of De Nicola and Hennessy [32], which is defined thus:

$$p \sqsubseteq_{MUST} q \text{ if } \forall r \,.\, p \textit{ must pass } r \text{ implies } q \textit{ must pass } r$$

The MUST-preorder, like Morris one, is *contextual*: to prove that $p \sqsubseteq_{MUST} q$, a quantification over an *infinite* number of clients is required, and so the definition of the preorder does not entail an effective proof method. The solution to this problem is to devise an alternative (semantic) characterisation of the preorder $\sqsubseteq_{MUST}$, i.e. a preorder $\preccurlyeq_{ALT}$ endowed with a practical proof method and such that the equality $\preccurlyeq_{ALT} = \sqsubseteq_{MUST}$ is true. On the Internet message-passing is asynchronous, because so is the standard TCP transmission. Moreover also lock-freedom in shared-memory concurrency gives rise to asynchronous information-flow between programs. The practical importance of asynchrony motivates pursuing results about $\sqsubseteq_{MUST}$ in a setting where there is a fundamental asymmetry between blocking actions and non-blocking actions. This asymmetry substantially complicates reasoning on $\sqsubseteq_{MUST}$, and a concrete example of these complications the completeness result of Castellani and Hennessy [33], which we have proven false. The state of the art on this topic is presented by [27], where all proofs are machine-checked, and the implementation is open-source and available at [34].

Content and input The characterization is designed for Labelled Transition System, and for a programming language, we need a reduction semantic. Then the Harmony Lemma, that links LTS and STS, is needed. It was sketched in many references, like in [2] or in [3], due the redundancy of arguments. Thus, COQ seems perfect to put a final act on this mathematic folklore. Our contribution is to fully prove the Harmony Lemma for the CCS, ACCS, VCCS and VACCS. And for each asynchronous language, prove that they are instance of each classes from [27], to build solid basics to analyse the MUST-preorder in an asynchronous setting, in particular, that the Axioms for Outbuffered Agents are true, which depends [27].

The main document is here to define in one paper the CCS and the proofs we called earlier. In chapter 1, we recall the scientific background that is needed for the document. In chapter 2 and 3, we define CCS and ACCS, with the Harmony Lemma and the Axioms from [1] In chapter 4, we introduce the value passing for CCS and ACCS. Finally, in chapter 5, we redesigned the VCCS language to introduce the sequencing. The appendix is here to inform about the COQ implementation.

Chapter 1

Preliminaries

1.1 Relations

A n-relation between n sets ($S_i, i \in \{1, 2, ..., n\}$) is a subset $\mathcal{R}^{(n)}$ of $S_1 \times S_2 \times ... \times S_n$ ($\mathcal{R} \subseteq S_1 \times S_2 \times ... \times S_n$). We note $s\mathcal{R}s'$ for $(s, s') \in \mathcal{R}$. We said that $\mathcal{R}^{(n)}$ is a binary relation if $n = 2$.

Let's take $\mathcal{R}$ a binary relation ($\subseteq S \times S$).

$\mathcal{R}$ is a **equivalence** relation if it satisfies all of this three properties :

$$\mathcal{R} \text{ is a } \textbf{reflexive} \text{ relation if for all } s \in S, s\mathcal{R}s. \tag{1.1}$$

$$\mathcal{R} \text{ is a } \textbf{symmetric} \text{ relation if for all } (s, s') \in S \times S, s\mathcal{R}s' \text{ implies } s'\mathcal{R}s. \tag{1.2}$$

$$\mathcal{R} \text{ is a } \textbf{transitive} \text{ relation if for all } (s, s', s'') \in (S \times S \times S), s\mathcal{R}s' \text{ and } s'\mathcal{R}s'' \text{ implies } s\mathcal{R}s''. \tag{1.3}$$

Reflexive closure of $\mathcal{R}$ is the least relation that contains $\mathcal{R}$ and that is reflexive.

Symmetric closure of $\mathcal{R}$ is the least relation that contains $\mathcal{R}$ and that is symmetric.

Transitive closure of $\mathcal{R}$ is the least relation that contains $\mathcal{R}$ and that is transitive.

Any of this closure can be viewed as completion of $\mathcal{R}$ to obtain reflexivity, symmetry or transitivity.

Let's take R_1 another binary relation, then for all s_1 and s_2, the relation $s_1 R.R_1 s_2$ is defined as , it exists some s' such that $s_1 R s'$ and $s' R_1 s_2$.

1.2 STS

Let $\mathcal{S}$ be the set of **States** of a system.
A **State Transition System** (or reduction semantic) is a binary relation $\mathcal{R}$ ($\subseteq S \times S$).
When $(s, t) \in \mathcal{R}$, we call it a **step**, and we write :

$$s \longrightarrow t$$

We note $\longrightarrow^*$ (resp. $\longrightarrow_=$) the transitive (resp. reflexive) closure of $\longrightarrow$.

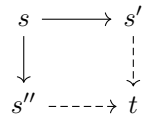
Example (Dobble game): Let's take 7 families of cards with 2 parents, 1 child and 1 aunt such that we have a set of 28 cards.

The goal of this game is to discard all of your cards.

We distribute the 28 cards between the players then we have 3 rules :

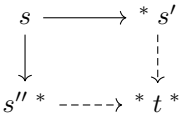
- we start our turn by taking one card from our right player
- then if we have 2 cards from the same family then we can discard it, we can repeat this action
- if no more actions can be made, we pass our turn

The reduction has the **diamond property** if for all s, $s \longrightarrow s'$ and $s \longrightarrow s''$, it exists t such that $s' \longrightarrow t$ and $s'' \longrightarrow t$.



Diamond property

The reduction has the **confluence property** if for all s, $s \longrightarrow^* s'$ and $s \longrightarrow^* s''$, it exists t such that $s' \longrightarrow^* t$ and $s'' \longrightarrow^* t$.



Confluence property

Remark: The diamond property implies the confluence.

Example (λ-calculus): The lambda calculus Λ is the terms defined by the following grammar :

$$P, P_1, P_2 := x \mid \lambda x.P \mid P_1\, P_2$$

and by the the following reduction (under any term):

$$(\lambda x.P_1)P_2 \longrightarrow P_1[x := P_2] \text{ (the substitution of all } x \text{ in } P_1 \text{ by } P_2)$$

Théorème 1.2.1 (Church-Rosser). *La relation $\longrightarrow$ est confluente.*

1.3 LTS

Let $\mathcal{S}$ be the set of **States** of a system, and let $\mathcal{A}$ be an alphabet of **Actions**.
A **Labelled Transition System** (LTS) on S consists of a relation :

$$\mathcal{R} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$$

When $(s, \alpha, t) \in \mathcal{R}$, we write :

$$s \xrightarrow{\alpha} t$$

We call s the **source** and t the **target**.
We note $s \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2$ for $s \xrightarrow{\alpha_1} s_1 \wedge s_1 \xrightarrow{\alpha_2} s_2$, this can be generalized for any length. If $s \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} s_3 \xrightarrow{\alpha_4} \ldots \xrightarrow{\alpha_n} s_n$ ($n \in \mathbb{N}$), then we call s_n a **derivative** of s under the **trace** $\alpha_1\alpha_2\alpha_3\alpha_4\ldots\alpha_n$.
An LTS can be thought of an automaton without a start state or accepting states.

If, for any given s and α, there exists only a single tuple $s \xrightarrow{\alpha} s'$ in $\mathcal{R}$ then one says that α is deterministic (for s).
If, for any given s and α, there exists at least one tuple $s \xrightarrow{\alpha} s'$ in $\mathcal{R}$, then one says that α is executable (for s)

or s can performs α.

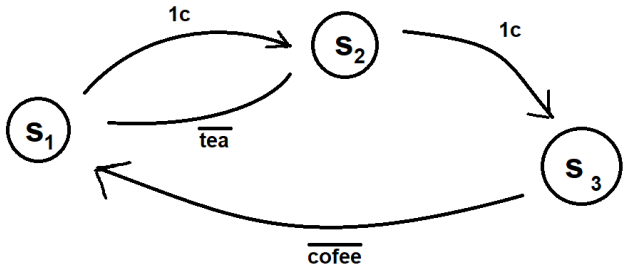
Let's take an infinite set $\mathcal{N}$ of **names**, we shall usually denote them with letter, but we can use words too.

Then we introduce $\overline{\mathcal{N}} := \{\overline{a} \mid a \in \mathcal{N}\}$, it's the **co-Actions**.
Generally we take $\mathcal{A}$ as $\mathcal{N} \cup \overline{\mathcal{N}}$.

Example (Vending Machine): We can view the $\mathcal{A}$ in terms of receiving something (for $\mathcal{N}$) or giving something (for $\overline{\mathcal{N}}$).

Our machine can give *coffee* or *tea*, but it can receive *1 coins*. To buy a coffee, we must pay 2 coins and only 1 for a tea.
So let's take $\mathcal{A} := \{\overline{coffee}, \overline{tea}, 1c\}$.

We can describe our machine as follow, we note s_i ($i \in \{1, 2, 3\}$) for the states of our machine after each actions :



Example ($\Lambda^!$-calculus): Let's take the λ-calculus with some black boxes ($!\square$).
We have the $\Lambda^!$-calculus.
We have then not only the β-reduction but we have $\beta_!$-reduction too. We can include into Actions, surface (s) reductions (s) and non-surface ($\neg s$) reduction.
Let's take for example $\mathcal{A} := \{s, \neg s, \beta, \beta_!\}$ and the LTS $(\Lambda^!, \mathcal{A}, \Lambda^!)$. Then, we look at this term :

$$(\lambda y.y)(\lambda^! x.x!x)(\lambda^! x.x!x)((\lambda z.z)w)$$

And we can describe it at follows :

$$\begin{array}{l} (\lambda^! x.x!x)!(\lambda^! x.x!x)((\lambda z.z)w) \xrightarrow{\beta} (\lambda^! x.x!x)!(\lambda^! x.x!x)w \xrightarrow{\beta_!} (\lambda^! x.x!x)!(\lambda^! x.x!x)w \\ \qquad\qquad \downarrow \beta_! \\ (\lambda^! x.x!x)!(\lambda^! x.x!x)((\lambda z.z)w) \end{array}$$

Example (Buffer): Let's take the set $\mathcal{S} := \{0, 1\}^*$ (the set of all finite sequences made with 0 and 1).
We take $\mathcal{A} := \{in_i \mid i \in \{0, 1\}\} \cup \{out_i \mid i \in \{0, 1\}\}$. And we describe the LTS $(\mathcal{S}, \mathcal{A}, \mathcal{S})$ as (for any $s \in \mathcal{S}$):

$$\frac{}{s \xrightarrow{in_0} s\,0} \quad \frac{}{s\,0 \xrightarrow{out_0} s}$$

$$\frac{}{s \xrightarrow{in_1} s\,1} \quad \frac{}{s\,1 \xrightarrow{out_1} s}$$

For $i \in \{0,1\}$, in_i puts i at the top of the sequence s, quite the opposite, out_i removes i from the sequence s. The action out_i (for any $i \in \{0,1\}$) requires a certain model to be performed. And any sequence can perform in_i (for any $i \in \{0,1\}$).
For instance, the sequence 1010 can't performs the action out_1. But it can performs out_0. And 111 is a derivative of 1010 :

$$1010 \xrightarrow{out_0} 101 \xrightarrow{out_1} 10 \xrightarrow{out_0} 1 \xrightarrow{in_1} 11 \xrightarrow{in_1} 111$$

Chapter 2

Calculus of Communicating Systems (CCS)

In this chapter, we will see the Calculus of Communicating Systems (CCS) introduced by Robin Milner in 1980 [2] with few differences :

- we didn't included the restriction of a Channel
- we added a recursion operator, like Hennessy [12] we kept the Success Process from the Characterization [27] (predicate GOOD)

We will cover his reduction semantic and transition system, and finally the link between those point of view : the Harmony Lemma.

2.1 Definition

The principle of CCS is that we have parallel processes that can communicate through Channels. Let's take a countable set as channels. We call it $\mathcal{C}$.

Definition 2.1.1 (Processes, Guards and ActionPrefixes): The processes $\mathcal{P}_{\text{CCS}}$ are :

(Process)
$$P, P_1, P_2 ::= P_1 \parallel P_2 \mid X \mid \text{rec}\, X.P \mid G$$

(Guards)
$$G, G_1, G_2 ::= ⓪ \mid ① \mid \pi \bullet P \mid G_1 + G_2$$

(ActionPrefix)
$$\pi ::= c? \mid c! \mid \tau \quad (c \in \mathcal{C})$$

$P_1 \parallel P_2$ is the parallel of two process, P_1 and P_2. The capital X is the variables designed for the recursion operator, it is destined to be replaced with a process. The recursion process rec $X.P$ is the process that loop it self.

Guards are special processes. In this family of processes, we have the inactive process : ⓪, the process that does nothing, passing to the next one.

The Success Process, ①, is only here to notify that a user is happy, or in a good mood (it will be used in future works).

We have the summation of a guarded process G_1 and G_2, $G_1 + G_2$. It is a process that offers the behaviors of all his guards, like the parallele. But it will discard all of the unused guards afterwards.

And finally, the process with an action prefix, $\pi \bullet P$. It is like the sequencing of actions : it will make some action π, and then behaves like P.

In terms of actions, we have three kind :

- the tau prefix : τ
- the input prefix : $c\,?$
- the output prefix : $c\,!$

The action τ is here to notify that there is an action that can be observed.

The action $c\,?$ is here to notify that the process must receive an information before continuing his computation. The name c is an identifier. The information is passing via the channel c.

The action $c\,!$ is here to notify that the process must send an information before continuing his computation. It is the counter part of the action $c\,?$, it is his co-action.

2.2 Congruence and reduction

Now we have a grammar for our terms (Definition 2.1.1), we want to mix up some process. To say, that they behave similarly for our reduction semantic.

First of all, let's define this relation, the structural congruence. To do that, we will need the definition of Context.

The Context is here to make the relation passing through the syntax of our grammar.

Definition 2.2.1 (Context for $\mathcal{P}_{\text{CCS}}$): A Context is a Process, following the Definition 2.1.1, with one or multiple hole(s) (noted $(\!|\,|\!)$).

We note $C(\!|P|\!)$, where the hole(s) in C are substituted by P.

Note, that a hole is in place of guards (like in the summation), then it can only be replaced by a guard G.

Example: Let's take $C = X \parallel (X + \textcircled{1})$.
Then the X's can only be replace by a guard.

Something like :

$$\times\ C(\!|\text{rec}\, X.\textcircled{0}|\!) := \text{rec}\, X.\textcircled{0} \parallel ((\text{rec}\, X.\textcircled{0}) + \textcircled{1})$$

is impossible.

That will produce : $(\text{rec}\, X.\textcircled{0}) + \textcircled{1}$. This wouldn't have any sense with our grammar ($\text{rec}\, X.\textcircled{0}$ isn't a guard).

Example: Let's take $C' = (\!|\,|\!) \parallel c\,? \bullet \textcircled{1} \parallel (\!|\,|\!)$.

Then, by definition :

$$\checkmark\ C'(\!|\textcircled{0}|\!) = \textcircled{0} \parallel c\,? \textcircled{1} \bullet \| \textcircled{0}$$

Definition 2.2.2 (Structural congruence for $\mathcal{P}_{\text{CCS}}$): The Structural Congruence over $\mathcal{P}_{\text{CCS}}$ is the reflexive, symmetric and transitive closure of the binary relation induced by the rules in figure 2.1.

In other terms, $(\mathcal{P}_{\text{CCS}}, +, \textcircled{0})$ and $(\mathcal{P}_{\text{CCS}}, \|, \textcircled{0})$ are commutative monoid. And the relation is still viable through any viable context, defined in the Definition 2.2.1.

$$
\begin{array}{rl}
\text{[Sc-sneut]} & G + \textcircled{0} \equiv G \\
\text{[Sc-scom]} & G_1 + G_2 \equiv G_2 + G_1 \\
\text{[Sc-sass]} & G_1 + (G_2 + G_3) \equiv (G_1 + G_2) + G_3 \\
\\
\text{[Sc-pneut]} & P \parallel \textcircled{0} \equiv P \\
\text{[Sc-pcom]} & P \parallel Q \equiv Q \parallel P \\
\text{[Sc-pass]} & P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \\
\\
\text{[Sc-context]} & C(\!|P|\!) \equiv C(\!|Q|\!) \qquad\qquad if P \equiv Q
\end{array}
$$

Figure 2.1: Structural congruence rules for CCS

Now, let's look the reduction semantic of CCS :

Definition 2.2.3 (Reduction semantic for CCS): The Reduction Semantic $\longrightarrow$ is the binary relation over $\mathcal{P}_{\text{CCS}}$ defined in figure 2.2. It defines the STS $(\mathcal{P}_{\text{CCS}}, \longrightarrow)$.

$$
\begin{array}{ll}
\text{[Sts-com]} \ \dfrac{}{c\,! \bullet P_1 + G_1 \parallel c\,? \bullet P_2 + G_2 \longrightarrow P_1 \parallel P_2} & \text{[Sts-tau]} \ \dfrac{}{\tau \bullet P + G \longrightarrow P} \\
\\
\text{[Sts-unf]} \ \dfrac{}{\text{rec}\, X.C(\!|X|\!) \longrightarrow C(\!|\text{rec}\, X.C(\!|X|\!)|\!)} & \text{[Sts-par]} \ \dfrac{P_1 \longrightarrow P_2}{P_1 \parallel Q \longrightarrow P_2 \parallel Q} \\
\\
\text{[Sts-cong]} \ \dfrac{P_1 \equiv Q_1 \quad Q_1 \longrightarrow Q_2 \quad Q_2 \equiv P_2}{P_1 \longrightarrow P_2} &
\end{array}
$$

Figure 2.2: The Reduction Semantic of $\mathcal{P}_{\text{ACCS}}$
The meta-variables are $c \in \mathcal{C}$.

Let's investigate a bit more about this Reduction.

The rule [Sts-par] means that if I do a computation, then I can do it in parallel of any process Q.

The rule [Sts-com] means that two processes in parallel can communicate through the channel c.

The rule [Sts-unf] is here for the recursion through the context C.

Example: Take the process P :

$$P := \text{rec}\, X.X \parallel X \parallel c\,? \bullet \textcircled{0}$$

Then, $P = \text{rec}\, X.C(\!|X|\!)$ with $C := (\!|\,|\!) \parallel (\!|\,|\!) \parallel c\,? \bullet \textcircled{0}$.

And we have :

$$\text{rec}\, X.X \parallel X \parallel c\,? \bullet \textcircled{0} \longrightarrow P \parallel P \parallel c\,? \bullet \textcircled{0}$$

The rule [Sts-cong] is here assimilate some program with other through the structural congruence, to force them to compute similarly.

Example: Take the process $P := a\,! \bullet \textcircled{1} \parallel a\,? \bullet \textcircled{1}$, we have :

$$a\,! \bullet \textcircled{1} \parallel a\,? \bullet \textcircled{1} \parallel \textcircled{0} \longrightarrow \textcircled{1}$$

Indeed :

$$\begin{aligned} a\,!\bullet\textcircled{1} \parallel a\,?\bullet\textcircled{0} \parallel \textcircled{0} &\equiv a\,!\bullet\textcircled{1} \parallel a\,?\bullet\textcircled{0} \\ &\equiv a\,!\bullet\textcircled{1} + \textcircled{0} \parallel a\,?\bullet\textcircled{0} \\ &\equiv a\,!\bullet\textcircled{1} + \textcircled{0} \parallel a\,?\bullet\textcircled{0} + \textcircled{0} \end{aligned}$$

And :

$$\textcircled{1} \parallel \textcircled{0} \equiv \textcircled{1}$$

Then, by the rule [Sts-cong], we have our reduction.

Lemma 2.2.4 (ReductionShape): $\forall (P, Q) \in \mathcal{P}^2_{\text{CCS}}$, if $P \longrightarrow Q$ then one of this cases occurs :

- $P \equiv c\,! \bullet P_1 + G_1 \parallel c\,? \bullet P_2 + G_2 \parallel S$ and $Q \equiv P_1 \parallel P_2 \parallel S$ for some P_1, G_1, P_2, G_2, S
- $P \equiv \tau \bullet P_1 + G_1 \parallel S$ and $Q \equiv P_1 \parallel S$ for some P_1, G_1, S
- $P \equiv \text{rec}\, X.C(\!|X|\!) \parallel S$ and $Q \equiv C(\!|\text{rec}\, X.C(\!|X|\!)|\!) \parallel S$ for some C, S

Remark: The lemma 2.2.4 reduce the proof that went to induction on the reduction.

Proof. The main strategy is to do an induction on the reduction $P \longrightarrow Q$.

- If the reduction was made by [Sts-com], [Sts-tau] or [Sts-unf], then we conclude by taking $S := \textcircled{0}$ and using [Sc-pneut]and [Sc-pcom].
- If the reduction was made by [Sts-par], then we conclude by using [Sc-pass] and by Induction Hypothesis.
- If the reduction was made by [Sts-cong], then we can conclude by the transitivity of the Structural Congruence and by Induction Hypothesis.

□

Lemma 2.2.5 (SimplifiedTree): $\forall (P, Q) \in \mathcal{P}_{\text{CCS}}$, if $P \longrightarrow Q$, then there exists a proof tree of length 3 that deduce $P \longrightarrow Q$.

2.3 Transition

The Reduction Semantic only focus on communication. We now focus on how to describe the process more precisely with a Labelled Transition System.

Definition 2.3.1 (Actions): We define the Actions $\mathcal{A}$ as :

- (the input-action) $c\,?$
- (the output-action) $c\,!$
- (the tau-action) τ

$(c \in \mathcal{C})$

Remark: The tau-action are the observable action.

Remark: There is actually no difference between action prefixes and the labels (action). It will change for VCCS.

Definition 2.3.2 (LTS for $\mathcal{P}_{\text{CCS}}$): The LTS $(\mathcal{P}_{\text{CCS}}, \mathcal{A}, \xrightarrow{\alpha})$ is defined in figure 2.3.

There are many more rules in the Reduction Semantic. Indeed, we haven't a rule like [Sts-cong] to mix-up everything and amalgamate processes.

But there are still some link between the structural congruence and the LTS, as we see with the next three lemmas.

[Input] $\dfrac{}{c?\bullet P \xrightarrow{c?} P}$ [Output] $\dfrac{}{c!\bullet P \xrightarrow{c!} P}$

[Tau] $\dfrac{}{\tau \bullet P \xrightarrow{\tau} P}$ [Unf] $\dfrac{}{\text{rec}\, X.C(\!|X|\!) \xrightarrow{\tau} C(\!|\text{rec}\, X.P|\!)}$

[Sum-L] $\dfrac{G_1 \xrightarrow{\alpha} P_1}{G_1 + G \xrightarrow{\alpha} P_1}$ [Sum-R] $\dfrac{G_1 \xrightarrow{\alpha} P_1}{G + G_1 \xrightarrow{\alpha} P_1}$

[Par-L] $\dfrac{P_1 \xrightarrow{\alpha} P_2}{P_1 \parallel P \xrightarrow{\alpha} P_2 \parallel P}$ [Par-R] $\dfrac{P_1 \xrightarrow{\alpha} P_2}{P \parallel P_1 \xrightarrow{\alpha} P \parallel P_2}$

[Com-L] $\dfrac{P_1 \xrightarrow{c?} P_2 \quad Q_1 \xrightarrow{c!} Q_2}{Q_1 \parallel P_1 \xrightarrow{\tau} Q_2 \parallel P_2}$ [Com-R] $\dfrac{P_1 \xrightarrow{c?} P_2 \quad Q_1 \xrightarrow{c!} Q_2}{P_1 \parallel Q_1 \xrightarrow{\tau} P_2 \parallel Q_2}$

Figure 2.3: The LTS of CCS
The meta-variables are $c \in \mathcal{C}, \alpha \in \mathcal{A}$.

Lemma 2.3.3 (TransitionShapeForInput): $\forall (P, Q) \in \mathcal{P}_{\text{CCS}}$ and $c \in \mathcal{C}$, if $P \xrightarrow{c?} Q$, then $P \equiv (c? \bullet P_1 + G_1) \parallel S$, $Q \equiv P_1 \parallel S$ for some P_1, G_1, S.

If P is a guard then $S = \textcircled{0}$.

Proof. The main strategy is to go with an induction on $P \xrightarrow{c?} Q$.

Then 5 cases remain :

- if the transition was made by [Input] :

  $$\frac{}{c? \bullet P_1 \xrightarrow{c?} P_1}$$

  By [Sc-sneut] and [Sc-pneut], we have $c? \bullet P_1 \equiv c? \bullet P_1 + \textcircled{0} \parallel \textcircled{0}$ and $P_1 \equiv P_1 \parallel \textcircled{0}$.

- if the transition was made by [Par-L] :

  We have $P := P_1 \parallel P'$ for some P_1 and P' and for some P_2 :

  $$\frac{P_1 \xrightarrow{c?} P_2}{P_1 \parallel P' \xrightarrow{c?} P_2 \parallel P'}$$

  By induction hypothesis, we have for some P_1', G_1', S' :

  $$P_1 \equiv (c? \bullet P_1' + G_1') \parallel S'$$
  $$P_2 \equiv P_1' \parallel S'$$

Then by [Sc-context] and [Sc-pass], we have :

$$\begin{aligned}(P :=)P_1 \parallel P' &\equiv (c\,?\bullet P'_1 + G'_1 \parallel S') \parallel P' \\ &\equiv (c\,?\bullet P'_1 + G'_1) \parallel (S' \parallel P')\end{aligned}$$

And

$$\begin{aligned}(Q :=)P_2 \parallel P' &\equiv (P'_1 \parallel S') \parallel P' \\ &\equiv P'_1 \parallel (S' \parallel P')\end{aligned}$$

- if the transition was made by [Par-R] :

  We have $P := P_1 \parallel P'$ for some P_1 and P' and for some P_2 :

$$\frac{P_1 \xrightarrow{c\,?} P_2}{P' \parallel P_1 \xrightarrow{c\,?} P' \parallel P_1}$$

  By induction hypothesis, we have for some P'_1, G'_1, S' :

$$\begin{aligned}P_1 &\equiv (c\,?\bullet P'_1 + G'_1) \parallel S' \\ P_2 &\equiv P'_1 \parallel S'\end{aligned}$$

  Then by [Sc-context], [Sc-pass] and [Sc-pcom], we have :

$$\begin{aligned}(P :=)P' \parallel P_1 &\equiv P' \parallel (c\,?\bullet P'_1 + G'_1 \parallel S') \\ &\equiv P' \parallel (S' \parallel c\,?\bullet P'_1 + G'_1) \\ &\equiv (P' \parallel S') \parallel (c\,?\bullet P'_1 + G'_1) \\ &\equiv (c\,?\bullet P'_1 + G'_1) \parallel (P' \parallel S')\end{aligned}$$

  And

$$\begin{aligned}(Q :=)P' \parallel P_2 &\equiv P' \parallel (P'_1 \parallel S') \\ &\equiv P' \parallel (S' \parallel P'_1) \\ &\equiv (P' \parallel S') \parallel P'_1 \\ &\equiv P'_1 \parallel (P' \parallel S')\end{aligned}$$

- if the transition was made by [Sum-L] :

  We have $P := G_1 + G'$ for some G_1 and G' :

$$\frac{G_1 \xrightarrow{c\,?} Q}{G_1 + G' \xrightarrow{c\,?} Q}$$

By induction hypothesis, we have for some P'_1, G'_1 :

$$G_1 \equiv (c\,?\bullet P'_1 + G'_1) \parallel \textcircled{0} \equiv c\,?\bullet P'_1 + G'_1$$
$$Q \equiv P'_1 \parallel \textcircled{0}$$

Then by [Sc-context], [Sc-sass] and [Sc-pneut], we have :

$$\begin{aligned}(P :=) G_1 + G' &\equiv (c\,?\bullet P'_1 + G'_1) + G' \\ &\equiv c\,?\bullet P'_1 + (G'_1 + G') \\ &\equiv c\,?\bullet P'_1 + (G'_1 + G') \parallel \textcircled{0}\end{aligned}$$

■ if the transition was made by [Sum-R] :

We have $P := G' + G_1$ for some G_1 and G' :

$$\frac{G_1 \xrightarrow{c\,?} Q}{G' + G_1 \xrightarrow{c\,?} Q}$$

By induction hypothesis, we have for some P'_1, G'_1 :

$$G_1 \equiv (c\,?\bullet P'_1 + G'_1) \parallel \textcircled{0} \equiv c\,?\bullet P'_1 + G'_1$$
$$Q \equiv P'_1 \parallel \textcircled{0}$$

Then by [Sc-context], [Sc-scom] , [Sc-sass] and [Sc-pneut], we have :

$$\begin{aligned}(P :=) G' + G_1 &\equiv G' + (c\,?\bullet P'_1 + G'_1) \\ &\equiv G' + (G'_1 + c\,?\bullet P'_1) \\ &\equiv (G' + G'_1) + c\,?\bullet P'_1 \\ &\equiv c\,?\bullet P'_1 + (G' + G'_1) \\ &\equiv c\,?\bullet P'_1 + (G' + G'_1) \parallel \textcircled{0}\end{aligned}$$

□

Lemma 2.3.4 (TransitionShapeForOutput): $\forall (P, Q) \in \mathcal{P}_{\text{CCS}}$ and $c \in \mathcal{C}$, if $P \xrightarrow{c\,!} Q$, then $P \equiv c\,!\bullet P_1 + G_1 \parallel S$, $Q \equiv P_1 \parallel S$ for some P_1, G_1, S.

If P is a guard then $S = \textcircled{0}$.

Proof. It is the same proof as for lemma 2.3.3.

We have just to replace the action $c\,?$ by the action $c\,!$. □

Lemma 2.3.5 (TransitionShapeForTauAndGuard): $\forall (P, Q) \in \mathcal{P}_{\text{CCS}}$ and $c \in \mathcal{C}$, if P is a guard and $P \xrightarrow{\tau} Q$, then $P \equiv \tau \bullet P_1 + G_1$, $Q \equiv P_1$ for some P_1, G_1.

Proof. The main strategy is to go with an induction on $P \xrightarrow{c\,?} Q$.

Then 3 cases remain :

- if the transition was made by [Tau] :

$$\frac{}{\tau \bullet Q \xrightarrow{\tau} Q}$$

By [Sc-sneut], we have $\tau \bullet Q \equiv \tau \bullet Q + \textcircled{0}$.

- if the transition was made by [Sum-L] :

$$\frac{G_1 \xrightarrow{\alpha} Q}{G_1 + G' \xrightarrow{\alpha} Q}$$

By induction hypothesis, we have for some G_1' :

$$G_1 \equiv \tau \bullet Q + G_1'$$

Then by [Sc-context] and [Sc-sass], we have :

$$\begin{aligned}(P :=) G_1 + G' &\equiv (\tau \bullet Q + G_1') + G' \\ &\equiv \tau \bullet Q + (G_1' + G')\end{aligned}$$

- if the transition was made by [Sum-R] :

$$\frac{G_1 \xrightarrow{\alpha} Q}{G' + G_1 \xrightarrow{\alpha} Q}$$

By induction hypothesis, we have for some G_1' :

$$G_1 \equiv \tau \bullet Q + G_1'$$

Then by [Sc-context], [Sc-scom] and [Sc-sass], we have :

$$\begin{aligned}(P :=) G' + G_1 &\equiv G' + (\tau \bullet Q + G_1') \\ &\equiv G' + (G_1' + \tau \bullet Q) \\ &\equiv G' + G_1' + \tau \bullet Q \\ &\equiv \tau \bullet Q + (G' + G_1')\end{aligned}$$

□

2.4 Harmony Lemma

Before talking about the Harmony Lemma, the link between the reduction semantic and the transition system, one important property is needed.

The property that says the structural congruence is a bisimulation for the transition $\xrightarrow{\alpha}$ for any α..

Lemma 2.4.1 (CongruenceRespectsTransition): $\forall (S, T) \in \mathcal{P}^2_{\text{CCS}},\ \alpha \in \mathcal{A}$, if $S \equiv . \xrightarrow{\alpha} T$ then $S \xrightarrow{\alpha} . \equiv T$.

Proof. Assume that $S \equiv T' \xrightarrow{\alpha} T$ for some T'.

The main strategy is to go with an induction on the length of $S \equiv T'$.

If the reduction is of length 1, then (because the structural congruence is an equivalence relation), it can be 3 cases :

- the reflexivity
- the rules in figure 2.1
- the symmetry of the rules in figure 2.1

■ for the reflexivity case, we have by hypothesis :

$$S \equiv S \xrightarrow{\alpha} T$$

Then by reflexivity for T, we have :

$$S \xrightarrow{\alpha} T \equiv T$$

■ for [Sc-pneut], we have by hypothesis :

$$P \parallel ⓪ \equiv P \xrightarrow{\alpha} T$$

Then by [Par-L] with $P \xrightarrow{\alpha} T$ and [Sc-pneut] we have :

$$P \parallel ⓪ \xrightarrow{\alpha} T \parallel ⓪ \equiv T$$

■ for the symmetry of [Sc-pneut], we have by hypothesis :

$$P \equiv P \parallel ⓪ \xrightarrow{\alpha} T$$

By construction, ⓪ makes no transition. Then the transition α was made by a [Par-L] rule.

Hence, it exists T' such that $P \xrightarrow{\alpha} T'$ and $T = T' \parallel ⓪$.

Then by the symmetry of [Sc-pneut] with T' we have :

$$P \xrightarrow{\alpha} T' \equiv T' \parallel ⓪ (:= T)$$

$$P \parallel ⓪ \xrightarrow{\alpha} T \parallel ⓪ \equiv T$$

■ for [Sc-pcom], we have by hypothesis :

$$P \parallel Q \equiv Q \parallel P \xrightarrow{\alpha} T$$

To go further, we have to analyse if the transition is made by P, Q or both (like for a [Com-L] rule).

Then we have 4 cases :

- if it was made by [Par-L], then it exists Q' such that $T = Q' \parallel P$ and :

$$Q \xrightarrow{\alpha} Q'$$

By [PAR-R] and [SC-PCOM], we have :

$$(S :=)P \parallel Q \xrightarrow{\alpha} P \parallel Q' \equiv Q' \parallel P(:= T)$$

- if it was made by [PAR-R], then it exists P' such that $T = Q \parallel P'$ and :

$$P \xrightarrow{\alpha} P'$$

By [PAR-L] and [SC-PCOM], we have :

$$(S :=)P \parallel Q \xrightarrow{\alpha} P' \parallel Q \equiv Q \parallel P'(:= T)$$

- if it was made by [COM-L], then it exists P' and Q' such that $T = Q' \parallel P'$ and :

$$P \xrightarrow{c!} P' Q \xrightarrow{c?} Q'$$

By [COM-R] and [SC-PCOM], we have :

$$(S :=)P \parallel Q \xrightarrow{\alpha} P' \parallel Q' \equiv Q' \parallel P'(:= T)$$

- if it was made by [COM-R], then it exists P' and Q' such that $T = Q' \parallel P'$ and :

$$P \xrightarrow{c?} P' Q \xrightarrow{c!} Q'$$

By [COM-L] and [SC-PCOM], we have :

$$(S :=)P \parallel Q \xrightarrow{\alpha} P' \parallel Q' \equiv Q' \parallel P'(:= T)$$

■ for [SC-PASS], we have by hypothesis :

$$(P \parallel Q) \parallel R \equiv P \parallel (Q \parallel R) \xrightarrow{\alpha} T$$

Again, we have to analyse if the transition is made by P or Q, or both (like for a [COM-L] rule).

Then we have 4 cases :

- if it was made by [COM-L], then it exists P' such that :

$$P \xrightarrow{c?} P'$$

But then, we have to determine which process between Q and R made the $c!$ transition.

1. if for some Q' :

$$Q \xrightarrow{c!} Q'$$

Then by [Com-L] :

$$P \parallel Q \xrightarrow{\tau} P' \parallel Q'$$

And by [Par-L] and [Sc-pass]:

$$(S :=)(P \parallel Q) \parallel R \xrightarrow{\tau} (P' \parallel Q') \parallel R \equiv P' \parallel (Q' \parallel R)(:= T)$$

2. if for some R' :

$$R \xrightarrow{c!} R'$$

Then by [Par-L] :

$$P \parallel Q \xrightarrow{c?} P' \parallel Q$$

And by [Com-L] and [Sc-pass]:

$$(S :=)(P \parallel Q) \parallel R \xrightarrow{\tau} (P' \parallel Q) \parallel R' \equiv P' \parallel Q \parallel R'(:= T)$$

- if it was made by [Com-R], then it exists P' such that :

$$P \xrightarrow{c!} P'$$

But then, we have to determine which process between Q and R made the $c?$ transition.

1. if for some Q' :

$$Q \xrightarrow{c?} Q'$$

Then by [Com-R] :

$$P \parallel Q \xrightarrow{\tau} P' \parallel Q'$$

And by [Par-L] and [Sc-pass]:

$$(S :=)(P \parallel Q) \parallel R \xrightarrow{\tau} (P' \parallel Q') \parallel R \equiv P' \parallel (Q' \parallel R)(:= T)$$

2. if for some R' :

$$R \xrightarrow{c?} R'$$

Then by [PAR-L] :

$$P \parallel Q \xrightarrow{c!} P' \parallel Q$$

And by [COM-R] and [SC-PASS]:

$$(S :=)(P \parallel Q) \parallel R \xrightarrow{\tau} (P' \parallel Q) \parallel R' \equiv P' \parallel Q \parallel R'(:= T)$$

- if it was made by [PAR-L], then it exists P' such that :

$$P \xrightarrow{\alpha} P'$$

Then by [PAR-L] and [SC-PASS], we have :

$$(P \parallel Q) \parallel R \xrightarrow{\alpha} (P' \parallel Q) \parallel R \equiv P \parallel Q \parallel R$$

- if it was made by [PAR-R], then we have for some T':

$$Q \parallel R \xrightarrow{\alpha} T'$$

But to go further, we have to analyse if it Q or R, or both, that made the transition.

1. if it was made by [COM-L], then for some Q', R', we have :

$$Q \xrightarrow{c?} Q' R \xrightarrow{c!} R'$$

By [PAR-R], we have :

$$P \parallel Q \xrightarrow{c?} P \parallel Q'$$

And by [COM-L] and [SC-PASS] we have :

$$(P \parallel Q) \parallel R \xrightarrow{\tau} (P \parallel Q') \parallel R' \equiv P \parallel (Q' \parallel R')$$

2. if it was made by [Com-R], then for some Q', R', we have :

$$Q \xrightarrow{c\,!} Q' R \xrightarrow{c\,?} R'$$

By [Par-R], we have :

$$P \parallel Q \xrightarrow{c\,!} P \parallel Q'$$

And by [Com-R] and [Sc-pass] we have :

$$(P \parallel Q) \parallel R \xrightarrow{\tau} (P \parallel Q') \parallel R' \equiv P \parallel (Q' \parallel R')$$

3. if it was made by [Par-L], then for some Q', we have :

$$Q \xrightarrow{\alpha} Q'$$

By [Par-R], we have :

$$P \parallel Q \xrightarrow{\alpha} P \parallel Q'$$

And by [Par-L] and [Sc-pass] we have :

$$(P \parallel Q) \parallel R \xrightarrow{\tau} (P \parallel Q') \parallel R \equiv P \parallel (Q' \parallel R)$$

4. if it was made by [Par-R], then for some R', we have :

$$R \xrightarrow{\alpha} R'$$

Then, by [Par-R] and [Sc-pass], we have :

$$(P \parallel Q) \parallel R \xrightarrow{\alpha} (P \parallel Q) \parallel R' \equiv P \parallel (Q \parallel R')$$

■ for [Sc-sneut], we have by hypothesis some G guard :

$$G + \textcircled{0} \equiv G \xrightarrow{\alpha} T$$

By [Sum-L] and the reflexivity, we have :

$$G + \textcircled{0} \xrightarrow{\alpha} T \equiv T$$

■ for the symmetry of [Sc-sneut], we have by hypothesis some G guard :

$$G \equiv G + ⓪ \xrightarrow{\alpha} T$$

⓪ makes no transition, then by construction, we have :

$$G \xrightarrow{\alpha} T$$

By reflexivity, we have :

$$G \xrightarrow{\alpha} T \equiv T$$

■ for [Sc-scom], we have by hypothesis some G guard :

$$G_1 + G_2 \equiv G_2 + G_1 \xrightarrow{\alpha} T$$

We have then to determine which of G_1 or G_2 makes the transition α.

- Assume that :

$$G_1 \xrightarrow{\alpha} T$$

Then, by [Sum-L] and reflexivity, we have :

$$G_1 + G_2 \xrightarrow{\alpha} T \equiv T$$

- Assume that :

$$G_2 \xrightarrow{\alpha} T$$

Then, by [Sum-R] and reflexivity, we have :

$$G_1 + G_2 \xrightarrow{\alpha} T \equiv T$$

■ for [Sc-sass], we have by hypothesis some G_1, G_2, G_3 guards such that :

$$G_1 + (G_2 + G_3) \equiv (G_1 + G_2) + G_3 \xrightarrow{\alpha} T$$

We have then to determine which of G_1, G_2 or G_3 makes the transition α.

- Assume that :

$$G_1 \xrightarrow{\alpha} T$$

Then, by [SUM-L] and reflexivity, we have :

$$G_1 + G_2 + G_3 \xrightarrow{\alpha} T \equiv T$$

- Assume that :

$$G_2 \xrightarrow{\alpha} T$$

Then, by [SUM-L], we have :

$$G_2 + G_3 \xrightarrow{\alpha} T$$

And finally, by [SUM-R] and reflexivity, we have :

$$G_1 + G_2 + G_3 \xrightarrow{\alpha} T \equiv T$$

- Assume that :

$$G_3 \xrightarrow{\alpha} T$$

Then, by [SUM-R], we have :

$$G_2 + G_3 \xrightarrow{\alpha} T$$

And finally, by [SUM-R] and reflexivity, we have :

$$G_1 + G_2 + G_3 \xrightarrow{\alpha} T \equiv T$$

■ for [SC-CONTEXT], we have by hypothesis some C context and P, Q such that :

$$P \equiv Q$$
$$C(\!|P|\!) \equiv C(\!|Q|\!) \xrightarrow{\alpha} T$$

We have then 3 cases :

- if the transition is make by C then we have immediately :

$$C(\!|P|\!) \xrightarrow{\alpha} T \equiv T$$

- if the transition is made by Q to Q', then $C(\!|Q'|\!) = T$ and by induction hypothesis it exists T' such that $P \xrightarrow{\alpha} T'$ and $T' \equiv Q'$.

  By [Sc-context], we have :

$$(S :=)C(\!|P|\!) \xrightarrow{\alpha} C(\!|T'|\!) \equiv C(\!|Q'|\!)(:= T)$$

- if the transition is made by Q with some process in C, we have $\alpha = \tau$ by communication. Then, we have 2 cases :

  1. if, we have for some Q', $T = C(\!|Q'|\!)$ and :

$$Q \xrightarrow{c?} Q'$$

     By induction hypothesis, we have for some P' :

$$P \xrightarrow{c?} P' \equiv Q'$$

     Then by [Com-L] or [Com-R], depending on the structure of C, we have :

$$(S :=)C(\!|P|\!) \xrightarrow{\tau} C(\!|P'|\!) \equiv C(\!|Q'|\!)(:= T)$$

  2. if, we have for some Q', $T = C(\!|Q'|\!)$ and :

$$Q \xrightarrow{c!} Q'$$

     By induction hypothesis, we have for some P' :

$$P \xrightarrow{c!} P' \equiv Q'$$

     Then by [Com-L] or [Com-R], depending on the structure of C, we have :

$$(S :=)C(\!|P|\!) \xrightarrow{\tau} C(\!|P'|\!) \equiv C(\!|Q'|\!)(:= T)$$

The proof is done for length 1 of the structural congruence.

Transitivity remains to be proved :

By hypothesis, we have for some S, S' :

$$S \equiv S' \equiv S'' \xrightarrow{\alpha} T$$

By induction hypothesis, it exists T' such that :

$$S' \xrightarrow{\alpha} T' \equiv T$$

Then by induction hypothesis, it exists T'' such that :

$$S \xrightarrow{\alpha} T'' \equiv T'(\equiv T)$$

By transitivity, we have :

$$S \xrightarrow{\alpha} T'' \equiv T$$

□

LEMMA 2.4.2 (Harmony Lemma): For all $(S,T) \in \mathcal{P}^2_{\text{CCS}}$, $S \longrightarrow T$ if and only if $S \xrightarrow{\tau} . \equiv T$.

Proof. We have two side to prove :

- if $S \longrightarrow T$ then $S \xrightarrow{\tau} . \equiv T$
- if $S \xrightarrow{\tau} . \equiv T$ then $S \longrightarrow T$

In the first part, the key lemmas are the lemma 2.4.1 and the lemma 2.2.4.

In the second part, the key lemmas are the lemma 2.3.3, lemma 2.3.4 and the lemma 2.3.5 and we will prove something stronger.

Indeed, if we have already that $S \xrightarrow{\tau} T$ implies $S \longrightarrow T$. Then we have the seconde part : Assume that $S \xrightarrow{\tau} . \equiv T$, then it exists S' such that :

$$S \xrightarrow{\tau} S'$$

Then we have $S \longrightarrow T$, by what we presupposed.

But, let's begin with the first part :

- Assume $S \longrightarrow T$.

  Then by lemma 2.2.4, there are 3 cases :

  - We have for some P_1, G_1, P_2, G_2, S :

$$P \equiv (c! \bullet P_1 + G_1 \parallel c? \bullet P_2 + G_2) \parallel S$$
$$Q \equiv (P_1 \parallel P_2) \parallel S$$

  We have, by [OUTPUT],[SUM-L] :

$$c! \bullet P_1 + G_1 \xrightarrow{c!} P_1$$

  And by [INPUT],[SUM-L] :

$$c?\bullet P_1 + G_1 \xrightarrow{c?} P_1$$

Then by [Com-R] and [Par-L], we have :

$$(c!\bullet P_1 + G_1 \parallel c?\bullet P_2 + G_2) \parallel S \xrightarrow{\tau} (P_1 \parallel P_2) \parallel S$$

By lemma 2.4.1, we have that, for some P' :

$$P \xrightarrow{\tau} P' \equiv (P_1 \parallel P_2) \parallel S (\equiv Q)$$

By transitivity, we have that :

$$P \xrightarrow{\tau} P' \equiv Q$$

- We have for some P_1, G_1, S:

$$P \equiv (\tau \bullet P_1 + G_1) \parallel S$$
$$Q \equiv P_1 \parallel S$$

We have, by [Tau],[Sum-L] and [Par-L] :

$$(\tau \bullet P_1 + G_1) \parallel S \xrightarrow{\tau} P_1 \parallel S$$

By lemma 2.4.1, we have that, for some P' :

$$P \xrightarrow{\tau} P' \equiv P_1 \parallel S (\equiv Q)$$

By transitivity, we have that :

$$P \xrightarrow{\tau} P' \equiv Q$$

- We have for some C, S:

$$P \equiv \text{rec}\, X.C(\!|X|\!) \parallel S$$
$$Q \equiv C(\!|\text{rec}\, X.C(\!|X|\!)|\!) \parallel S$$

We have, by [Unf] and [Par-L] :

$$\text{rec}\, X.C(\!|X|\!) \parallel S \xrightarrow{\tau} C(\!|\text{rec}\, X.C(\!|X|\!)|\!) \parallel S$$

By lemma 2.4.1, we have that, for some P' :

$$P \xrightarrow{\tau} P' \equiv C(\!|\text{rec } X.C(\!|X|\!)|\!) \parallel S(\equiv Q)$$

By transitivity, we have that :

$$P \xrightarrow{\tau} P' \equiv Q$$

In the other side :

■ Assume $S \xrightarrow{\tau} T$.

Then we do an induction on $S \xrightarrow{\tau} T$.

- if it is from [Tau] then we have for some P :

$$\tau \bullet T \xrightarrow{\tau} T$$
$$S = \tau \bullet T$$

By [Sc-sneut] and [Sc-pneut], we have :

$$\tau \bullet T \equiv (\tau \bullet T + \textcircled{0}) \parallel \textcircled{0}$$

And by [Sts-cong] we have :

$$\tau \bullet T \longrightarrow T$$

- if it is from [Unf] then we have for some P :

$$\text{rec } X.C(\!|P|\!) \xrightarrow{\tau} C(\!|\text{rec } X.C(\!|P|\!)|\!)$$
$$S = \text{rec } X.C(\!|P|\!)$$

And by [Sts-unf] we have :

$$S \longrightarrow T$$

- if it is from [Com-L] then we have for some P_1, P_2, P_1', P_2' :

$$P_1 \xrightarrow{c\,?} P_1'$$
$$P_2 \xrightarrow{c\,!} P_2'$$
$$S = P_1 \parallel P_2$$
$$T = P_1' \parallel P_2'$$

By lemma 2.3.3, we have :

$$P_1 \equiv c\,?\bullet P_1'' + G_1 \parallel Q_1$$
$$P_1' \equiv P_1'' \parallel Q_1$$

By lemma 2.3.4, we have :

$$P_2 \equiv c\,?\bullet P_2'' + G_2 \parallel Q_2$$
$$P_2' \equiv P_2'' \parallel Q_2$$

By [Sts-com], [Sts-par] (with $Q_1 \parallel Q_2$) and [Sts-cong] we have :

$$P_2 \parallel P_1 \longrightarrow P_1' \parallel P_2'$$

- if it is from [Com-R] then we have for some P_1, P_2, P_1', P_2' :

$$P_1 \xrightarrow{c\,!} P_1'$$
$$P_2 \xrightarrow{c\,?} P_2'$$
$$S = P_1 \parallel P_2$$
$$T = P_1' \parallel P_2'$$

By lemma 2.3.3, we have :

$$P_1 \equiv c\,!\bullet P_1'' + G_1 \parallel Q_1$$
$$P_1' \equiv P_1'' \parallel Q_1$$

By lemma 2.3.4, we have :

$$P_2 \equiv c\,!\bullet P_2'' + G_2 \parallel Q_2$$
$$P_2' \equiv P_2'' \parallel Q_2$$

By [Sts-com], [Sts-par] (with $Q_1 \parallel Q_2$) and [Sts-cong] we have :

$$(S :=)P_2 \parallel P_1 \longrightarrow P_1' \parallel P_2'(:= T)$$

- if it is from [Par-L] then we have for some P_1, P_1', Q:

$$P_1 \xrightarrow{\tau} P_1'$$
$$S = P_1 \parallel Q$$
$$T = P_1' \parallel Q$$

By induction hypothesis and by [STS-PAR], we have that :

$$P_1 \parallel Q \longrightarrow P_1' \parallel Q$$

- if it is from [PAR-R] then we have for some P_1, P_1', Q :

$$\begin{aligned} &P_1 \xrightarrow{\tau} P_1' \\ &S = Q \parallel P_1 \\ &T = Q \parallel P_1' \end{aligned}$$

By induction hypothesis and by [STS-PAR] and [STS-CONG] (with [SC-PCOM]), we have that :

$$Q \parallel P_1 \longrightarrow Q \parallel P_1'$$

- if it is from [SUM-L] then we have for some G_1, G_2, P :

$$\begin{aligned} &G_1 \xrightarrow{\tau} T \\ &S = G_1 + G_2 \end{aligned}$$

By lemma 2.3.5, for some G, we have :

$$\begin{aligned} &G_1 \equiv \tau \bullet P + G \\ &T \equiv P \end{aligned}$$

By [STS-TAU] and [STS-CONG], we have :

$$G_1 + G_2 \longrightarrow T$$

- if it is from [SUM-R] then we have for some G_1, G_2, P :

$$\begin{aligned} &G_1 \xrightarrow{\tau} T \\ &S = G_2 + G_1 \end{aligned}$$

By lemma 2.3.5, for some G, we have :

$$\begin{aligned} &G_1 \equiv \tau \bullet P + G \\ &T \equiv P \end{aligned}$$

By [STS-TAU] and [STS-CONG] (with [SC-SCOM]), we have :

$$G_1 + G_2 \longrightarrow T$$

□

Chapter 3

Asynchronous CCS (ACCS)

This chapter will introduce, like in [28], Asynchronous Calculus of Communicating Systems (ACCS). Compared to CCS, ACCS is a sub language of it. The main differences are :

- the output process $c\,!\,\bullet P$ is no longer a guard
- in the output process $c\,!\,\bullet P$, $P = \textcircled{0}$

Finally, we will talk about the characterization of asynchrony through the axioms of Peter Selinger from [1].

3.1 Congruence, Reduction and Transition

As discussed in the introduction of this chapter, here is the ACCS grammar :

Definition 3.1.1 (Processes, Guards and ActionPrefixes): The proccesses $\mathcal{P}_{\text{ACCS}}$:

(Process)
$$P, P_1, P_2 ::= P_1 \parallel P_2 \mid X \mid \text{rec}\, X.P \mid c\,!\,\bullet\textcircled{0} \mid G$$

(Guards)
$$G, G_1, G_2 ::= \textcircled{0} \mid \textcircled{1} \mid c\,?\,\bullet P \mid \tau\,\bullet P \mid G_1 + G_2$$

(Action Prefix)
$$\pi ::= c\,?\ \mid c\,! \mid \tau$$

($c \in \mathcal{C}$)

Definition 3.1.2 (Context for $\mathcal{P}_{\text{CCS}}$): A Context is a Process, following the Definition 3.1.1, with one or multiple hole(s) (noted $(\!|)$).

The Structural Congruence remains the same as for CCS.

Definition 3.1.3 (Structural Congruence for $\mathcal{P}_{\text{ACCS}}$): The Strutural Congruence over $\mathcal{P}_{\text{ACCS}}$ is the reflexive, symetric and transitive closure of the binary relation induced by the rules in figure 3.1.

Definition 3.1.4 (Reduction Semantic for $\mathcal{P}_{\text{ACCS}}$): The Reduction Semantic $\longrightarrow$ over $\mathcal{P}_{\text{ACCS}}$ is defined in figure 3.2. It defines the STS $(\mathcal{P}_{\text{ACCS}}, \longrightarrow)$.

The main difference is the rule [Sts-com] :

$$\frac{}{c\,!\,\bullet\textcircled{0} \parallel c\,?\,\bullet P_2 + G_2 \longrightarrow \textcircled{0} \parallel P_2}$$

$$
\begin{array}{rl}
\text{[Sc-sneut]} & G + \textcircled{0} \equiv G \\
\text{[Sc-scom]} & G_1 + G_2 \equiv G_2 + G_1 \\
\text{[Sc-sass]} & G_1 + (G_2 + G_3) \equiv (G_1 + G_2) + G_3 \\
 & \\
\text{[Sc-pneut]} & P \parallel \textcircled{0} \equiv P \\
\text{[Sc-pcom]} & P \parallel Q \equiv Q \parallel P \\
\text{[Sc-pass]} & P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \\
 & \\
\text{[Sc-context]} & C(\!|P|\!) \equiv C(\!|Q|\!) \qquad\qquad if P \equiv Q
\end{array}
$$

Figure 3.1: Structural congruence rules for ACCS

$$
\text{[Sts-com]}\ \frac{}{c\,!\bullet\textcircled{0} \parallel c\,?\bullet P_2 + G_2 \longrightarrow \textcircled{0} \parallel P_2} \qquad \text{[Sts-tau]}\ \frac{}{\tau \bullet P + G \longrightarrow P}
$$

$$
\text{[Sts-unf]}\ \frac{}{\text{rec}\, X.C(\!|X|\!) \longrightarrow C(\!|\text{rec}\, X.C(\!|X|\!)|\!)} \qquad \text{[Sts-par]}\ \frac{P_1 \longrightarrow P_2}{P_1 \parallel Q \longrightarrow P_2 \parallel Q}
$$

$$
\text{[Sts-cong]}\ \frac{P_1 \equiv Q_1 \quad Q_1 \longrightarrow Q_2 \quad Q_2 \equiv P_2}{P_1 \longrightarrow P_2}
$$

Figure 3.2: The STS of ACCS
The meta-variables are $c \in \mathcal{C}$.

Indeed, compared to CCS :

$$
\frac{}{c\,!\bullet P_1 + G_1 \parallel c\,?\bullet P_2 + G_2 \longrightarrow P_1 \parallel P_2}
$$

The output process is no longer a guard, so a process like that :

$$
\times\ c\,!\bullet\textcircled{0} + c\,?\bullet\textcircled{1}
$$

is no longer allowed.

Definition 3.1.5 (LTS for $\mathcal{P}_{\text{ACCS}}$): The LTS $(\mathcal{P}_{\text{ACCS}}, \mathcal{A}, \xrightarrow{\alpha})$ is defined in figure 3.3.

The Transition System is the same as for CCS, the main precision is in the rule for the Output Transition. This is the Output Transition [Output] for CCS :

$$
\frac{}{c\,!\bullet P \xrightarrow{c\,!} P}
$$

And we restrict $c\,!\bullet P$ for only $P = \textcircled{0}$, we have exactly [Output] for ACCS :

$$
\frac{}{c\,!\bullet\textcircled{0} \xrightarrow{c\,!} \textcircled{0}}
$$

$$[\textsc{Input}]\ \frac{}{c?\bullet P \xrightarrow{c?} P} \qquad [\textsc{Output}]\ \frac{}{c!\bullet\textcircled{0} \xrightarrow{c!} \textcircled{0}}$$

$$[\textsc{Tau}]\ \frac{}{\tau\bullet P \xrightarrow{\tau} P} \qquad [\textsc{Unf}]\ \frac{}{\text{rec}\,X.C(\!|X|\!) \xrightarrow{\tau} C(\!|\text{rec}\,X.P|\!)}$$

$$[\textsc{Sum-L}]\ \frac{G_1 \xrightarrow{\alpha} P_1}{G_1 + G \xrightarrow{\alpha} P_1} \qquad [\textsc{Sum-R}]\ \frac{G_1 \xrightarrow{\alpha} P_1}{G + G_1 \xrightarrow{\alpha} P_1}$$

$$[\textsc{Par-L}]\ \frac{P_1 \xrightarrow{\alpha} P_2}{P_1 \parallel P \xrightarrow{\alpha} P_2 \parallel P} \qquad [\textsc{Par-R}]\ \frac{P_1 \xrightarrow{\alpha} P_2}{P \parallel P_1 \xrightarrow{\alpha} P \parallel P_2}$$

$$[\textsc{Com-L}]\ \frac{P_1 \xrightarrow{c?} P_2 \quad Q_1 \xrightarrow{c!} Q_2}{Q_1 \parallel P_1 \xrightarrow{\tau} Q_2 \parallel P_2} \qquad [\textsc{Com-R}]\ \frac{P_1 \xrightarrow{c?} P_2 \quad Q_1 \xrightarrow{c!} Q_2}{P_1 \parallel Q_1 \xrightarrow{\tau} P_2 \parallel Q_2}$$

Figure 3.3: The LTS of ACCS
The meta-variables are $c \in \mathcal{C}, \alpha \in \mathcal{A}$.

3.2 (A)CCS Properties

In this section, we will maintain the properties that we saw in the Chapter 2, like the Output Shape and the Harmony Lemma.

The proof will be skipped in this section, as it is the same scheme as CCS.

Lemma 3.2.1 (ReductionShape): $\forall (P,Q) \in \mathcal{P}^2_{\text{ACCS}}$, if $P \longrightarrow Q$ then one of this cases occurs :

- $P \equiv c!\bullet\textcircled{0} \parallel c?\bullet P_2 + G_2 \parallel S$ and $Q \equiv P_2 \parallel S$ for some P_2, G_2, S
- $P \equiv \tau\bullet P_1 + G_1 \parallel S$ and $Q \equiv P_1 \parallel S$ for some P_1, G_1, S
- $P \equiv \text{rec}\,X.C(\!|X|\!) \parallel S$ and $Q \equiv C(\!|\text{rec}\,X.C(\!|X|\!)|\!) \parallel S$ for some C, S

Proof. See proof of lemma 2.2.4. □

Compared to the lemma 2.2.4, the first sentence would be :

$$P \equiv c?\bullet P_2 + G_2 \parallel c!\bullet\textcircled{0} \parallel S \text{ and } Q \equiv \textcircled{0} \parallel P_2 \parallel S$$

It takes one more step for the Structural Congruence to have lemma 3.2.1 :

$$Q \equiv P_2 \parallel S$$

Lemma 3.2.2 (SimplifiedTree): $\forall (P,Q) \in \mathcal{P}_{\text{ACCS}}$, if $P \longrightarrow Q$, then there exists a proof tree of length 3 that deduce $P \longrightarrow Q$.

Lemma 3.2.3 (TransitionShapeForInput): $\forall (P,Q) \in \mathcal{P}_{\text{ACCS}}$ and $c \in \mathcal{C}$, if $P \xrightarrow{c?} Q$, then $P \equiv (c?\bullet P_1 + G_1) \parallel S$, $Q \equiv P_1 \parallel S$ for some P_1, G_1, S.

If P is a guard then $S = \textcircled{0}$.

Proof. See proof of lemma 2.3.3 □

Lemma 3.2.4 (TransitionShapeForOutput): $\forall (P,Q) \in \mathcal{P}_{\text{ACCS}}$ and $c \in \mathcal{C}$, if $P \xrightarrow{c!} Q$, then $P \equiv c!\bullet\textcircled{0} \parallel S$, $Q \equiv S$ for some S.

Proof. See proof of lemma 2.3.4 □

There are 2 points to notice, compared to the lemma 2.3.4 :

- The sentence *"If P is a guard then* $S = \textcircled{0}$*"* disappears.

  Indeed, it makes no more sense, since $c\,! \bullet \textcircled{0}$ is no longer a guard.

- The shape of Q is simplified as S, the parallel process that was in parallel of $c\,! \bullet \textcircled{0}$. Then we have a additional property :

$$\begin{aligned} P &\equiv c\,! \bullet \textcircled{0} \parallel S \\ &\equiv c\,! \bullet \textcircled{0} \parallel Q \ \text{(since } \ Q \equiv S \ \text{)} \end{aligned}$$

LEMMA 3.2.5 (TransitionShapeForTauAndGuard): $\forall (P, Q) \in \mathcal{P}_{\text{ACCS}}$ and $c \in \mathcal{C}$, if P is a guard and $P \xrightarrow{\tau} Q$, then $P \equiv \tau \bullet P_1 + G_1$, $Q \equiv P_1$ for some P_1, G_1.

Proof. See proof of lemma 2.3.5 □

LEMMA 3.2.6 (CongruenceRespectsTransition): $\forall (S, T) \in \mathcal{P}^2_{\text{ACCS}},\ \alpha \in \mathcal{A}$, if $S \equiv . \xrightarrow{\alpha} T$ then $S \xrightarrow{\alpha} . \equiv T$.

Proof. See proof of lemma 2.4.1 □

LEMMA 3.2.7 (Harmony Lemma): For all $(S, T) \in \mathcal{P}^2_{\text{CCS}}$, $S \longrightarrow T$ if and only if $S \xrightarrow{\tau} . \equiv T$.

Proof. See proof of lemma 2.4.2 □

3.3 Asynchronicity

Peter Selinger, in [1], studies properties of asynchronous communication independently of any concrete concurrent process. He gives a general-purpose, mathematically rigorous definition of several notions of asynchrony in a natural setting where an agent is asynchronous if its input and/or output is filtered through a buffer or a queue, possibly with feedback.

We recall in this section his Axioms and connect them to the characterization of [27].

In his paper, Peter Selinger gives 5 axioms for asynchrony for outputs :

- OUTPUT-COMMUTATIVITY (FB1)
- OUTPUT-CONFLUENCE (FB2)
- OUTPUT-DETERMINACY (FB3)
- FEEDBACK (FB4)
- OUTPUT-TAU (FB5)

OUTPUT-COMMUTATIVITY (FB1) says that the output action out x can commute with any action α in the STS. In other words, the output action can be delayed.

For the OUTPUT-CONFLUENCE (FB2), it describe the fact that any action α ($\neq$ out x and $\neq \tau$) can be computed in parallel for any output action.

OUTPUT-DETERMINACY (FB3) characterize processes that are targets for an output action.

The FEEDBACK (FB4) axiom says that an observable action τ can be made by process that does an input action in x and an output action out x.

Finally, OUTPUT-TAU (FB5) completes OUTPUT-CONFLUENCE (FB2) for the action $\alpha = \tau$.
It splits in two cases :

- τ by communication
- τ for any other observable action

Like Peter Selinger said in his article, their axioms are not independant.

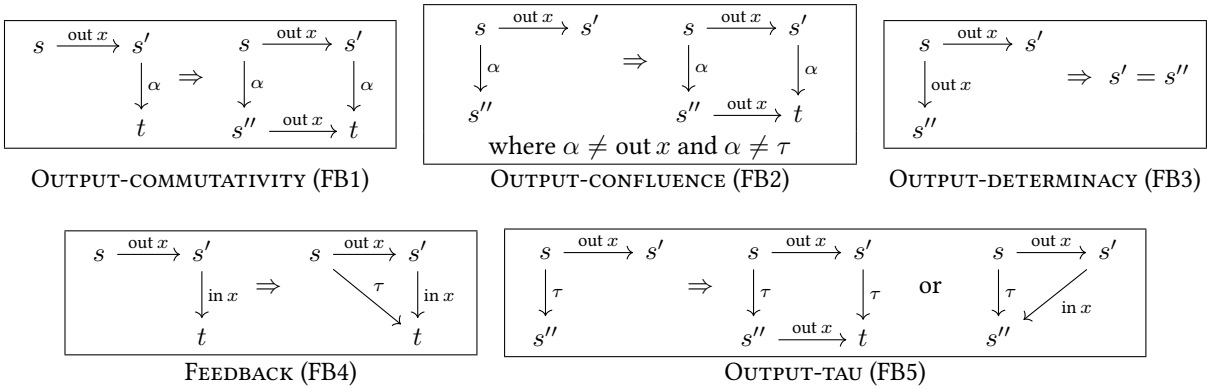


Figure 3.4: First-order axioms for out-buffered agents with feedback

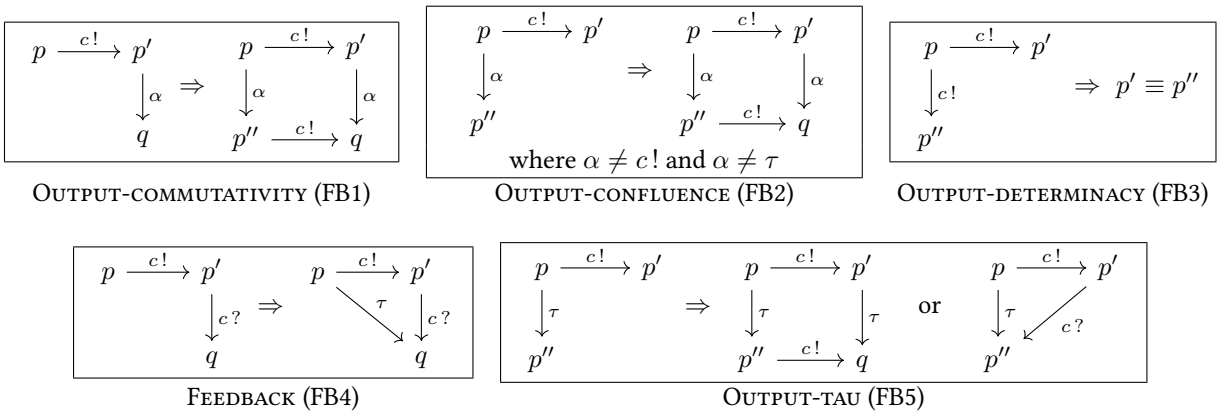
The axioms are made for a general purpose. Then we adapted them for ACCS.

An output action out x correspond to an action $x\,!$ and an output action in x correspond to an action $x\,?$

τ remains the same.

Finally, we are reasoning up to structural equivalence. With this in mind, the lemma 3.2.6 would be our main ally in the proofs.

With that in mind, this is the First-order axioms for out-buffered agents with feedback from Peter Selinger adapted to $ACCS$:



Remark: For CCS, the process Δ:

$$c\,! \bullet (c\,? \bullet \textcircled{1}) + c\,! \bullet \textcircled{1} + \tau \bullet \textcircled{0} + d\,! \bullet \textcircled{0}$$

makes a counter example of all this axioms.

Indeed :

- Output-commutativity (FB1)

  We have that :

$$\Delta \xrightarrow{c!} c?\bullet\textcircled{1}$$
$$\downarrow c?$$
$$\textcircled{1}$$

But $c!\bullet(c?\bullet\textcircled{1}) + c!\bullet\textcircled{1} + \tau\bullet\textcircled{0} + d!\bullet\textcircled{0}$ does not any Input Action $c?$.

- Output-confluence (FB2)

  We have that :

$$\Delta \xrightarrow{c!} c?\bullet\textcircled{1}$$
$$\downarrow d!$$
$$\textcircled{0}$$

  But $\textcircled{0}$ does no transition, especially no c ! action.

- Output-determinacy (FB3)

  We have that :

$$\Delta \xrightarrow{c!} \textcircled{1}$$
$$\downarrow c!$$
$$c?\bullet\textcircled{1}$$

  But $c?\bullet\textcircled{1}$ and $\textcircled{1}$ are not in the same equivalence class.

- Feedback (FB4)

  We have that :

$$\Delta \xrightarrow{c!} c?\bullet\textcircled{1}$$
$$\downarrow c?$$
$$\textcircled{1}$$

  But the only τ transition, that Δ does, is :

$$\Delta \xrightarrow{\tau} \textcircled{0}$$

  And $\textcircled{0}$ and $\textcircled{1}$ are not in the same equivalence class.

- Output-tau (FB5)

$$p \xrightarrow{c!} \textcircled{1}$$
$$\downarrow \tau$$
$$\textcircled{0}$$

  But $\textcircled{0}$ and $\textcircled{1}$ makes no transition.

Now, let's prove the axioms of Peter Selinger for ACCS.

Output-commutativity (FB1). Suppose that we have :

$$p \xrightarrow{c!} p'$$
$$\downarrow \alpha$$
$$q$$

Then by lemma 3.2.4 :

$$p \equiv c\,! \bullet \textcircled{0} \parallel p'$$

Then by [PAR-R] :

$$c\,! \bullet \textcircled{0} \parallel p' \xrightarrow{\alpha} c\,! \bullet \textcircled{0} \parallel q$$

And by lemma 3.2.6, there exists M such that :

$$p \xrightarrow{\alpha} M \equiv c\,! \bullet \textcircled{0} \parallel q$$

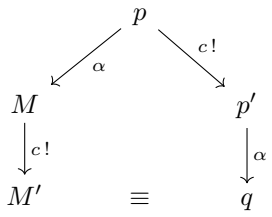
Remark, that :

$$M \equiv c\,! \bullet \textcircled{0} \parallel q \xrightarrow{c\,!} \textcircled{0} \parallel q$$

Then by lemma 3.2.6, there exists M' such that :

$$M \xrightarrow{c\,!} M' \equiv \textcircled{0} \parallel q \,(\equiv q)$$

Finally :



□

OUTPUT-CONFLUENCE (FB2). Suppose that we have :

$$\begin{array}{c} p \xrightarrow{c\,!} p' \\ \downarrow \alpha \\ p'' \end{array}$$
$$(\alpha \neq c\,! \text{ and } \alpha \neq \tau\ )$$

By lemma 3.2.4, we have :

$$c\,! \bullet \textcircled{0} \parallel p' \equiv p$$

Remark that :

$$c\,! \bullet \textcircled{0} \parallel p' \equiv p \xrightarrow{\alpha} p''$$

By the lemma 3.2.6, there exists M such that :

$$c\,! \bullet \textcircled{0} \parallel p' \xrightarrow{\alpha} M \equiv p''$$

Since α is different from τ and $c\,!$, then obviously, p' does the transition α to some M_2 and we have :

$$c\,! \bullet \textcircled{0} \parallel p' \xrightarrow{\alpha} c\,! \bullet \textcircled{0} \parallel M_2\ (:= M)$$

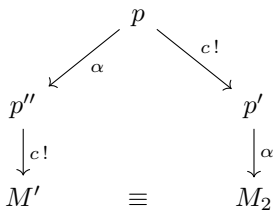
We have :

$$p'' \equiv c\,! \bullet \textcircled{0} \parallel M_2 \xrightarrow{c\,!} \textcircled{0} \parallel M_2$$

Then by the lemma 3.2.6, there exists M' such that :

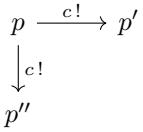
$$p'' \xrightarrow{c\,!} M' \equiv (\textcircled{0} \parallel M_2)\ (\equiv M_2)$$

Finally :



□

Output-determinacy (FB3). Suppose that we have :



We go with an induction on $p \xrightarrow{c\,!} p'$.

- by [Output], then :

  $$p = c\,! \bullet \textcircled{0}$$

  And $p' = p''$, hence $p' \equiv p''$.

- by [Par-L], then :

  $$p = P_1 \parallel P_2 \text{ and } P_1 \xrightarrow{c\,!} P_1'$$

  Remember that :

  $$(p =)\ P_1 \parallel P_2 \xrightarrow{c\,!} p''$$

  We have then 2 cases :

  - P_1 makes the action $c\,!$ to a P_1''
  - P_2 makes the action $c\,!$ to a P_2''

  In the first case, by induction hypothesis :

  $$P_1' \equiv P_1''$$

  And then :

  $$(P_1' \parallel P_2 =)\ p' \equiv p''\ (= P_1'' \parallel P_2)$$

  In the second case, we have :

  $$P_2 \xrightarrow{c\,!} P_2''$$

Then by lemma 3.2.4, we have :

$$P_2 \equiv c\,! \bullet \textcircled{0} \parallel P_2''$$
$$P_1 \equiv c\,! \bullet \textcircled{0} \parallel P_1'$$

Remember that we have to show that :

$$(p' =)\ P_1' \parallel P_2 \equiv P_1 \parallel P_2''\ (= p'')$$

Then by [Sc-context], we have :

$$P_1' \parallel P_2 \equiv P_1' \parallel c\,! \bullet \textcircled{0} \parallel P_2'' \equiv c\,! \bullet \textcircled{0} \parallel P_1' \parallel P_2'' \equiv P_1 \parallel P_2''$$

□

Feedback (FB4). Suppose that we have :

$$\begin{array}{ccc} p & \xrightarrow{c\,!} & p' \\ & & \downarrow{\scriptstyle c\,?} \\ & & q \end{array}$$

By lemma 3.2.4, we have :

$$p \equiv c\,! \bullet \textcircled{0} \parallel p'$$

Remark that :

$$c\,! \bullet \textcircled{0} \xrightarrow{c\,!} \textcircled{0} \text{ et } p' \xrightarrow{c\,?} q$$

Then by [Com-R] we have :

$$c\,! \bullet \textcircled{0} \parallel p' \xrightarrow{\tau} \textcircled{0} \parallel q$$

Then by [Com-R] we have :

$$p \equiv c\,! \bullet \textcircled{0} \parallel p' \xrightarrow{\tau} \textcircled{0} \parallel q$$

Then by lemma 3.2.6, there exists some q' such that :

$$p \xrightarrow{\tau} q' \equiv \textcircled{0} \parallel q\ (\equiv q)$$

Finally :

$$\begin{array}{ccc} p & \xrightarrow{\quad c\,! \quad} & p' \\ \downarrow{\scriptstyle \tau} & & \downarrow{\scriptstyle c\,?} \\ q' & \equiv & q \end{array}$$

□

Output-tau (FB5). Suppose that :

$$\begin{array}{ccc} p & \xrightarrow{c!} & p' \\ \downarrow \tau & & \\ p'' & & \end{array}$$

By lemma 3.2.4, we have :

$$c! \bullet \textcircled{0} \parallel p' \equiv p$$

Then by lemma 3.2.6, there exists M such that :

$$c! \bullet \textcircled{0} \parallel p' \xrightarrow{\tau} M \equiv p''$$

We have 2 cases for the τ action :

- it is made by only p'
- it is made by p' and $c! \bullet \textcircled{0}$, communication via c

In the first case, we have for some p_1:

$$p' \xrightarrow{\tau} p_1 \text{ and } M = c! \bullet \textcircled{0} \parallel p_1$$

And we have :

$$p'' \equiv c! \bullet \textcircled{0} \parallel p_1 \xrightarrow{c!} \textcircled{0} \parallel p_1$$

So by lemma 3.2.6, there exists M' such that :

$$p'' \xrightarrow{c!} M' \equiv \textcircled{0} \parallel p_1 \ (\equiv p_1)$$

Finally :

$$\begin{array}{ccccc} & & p & & \\ & \swarrow^{\tau} & & \searrow^{c!} & \\ p'' & & & & p' \\ \downarrow c! & & & & \downarrow \tau \\ M & & \equiv & & p_1 \end{array}$$

In the second case, the action τ is made by p' and $c! \bullet \textcircled{0}$ via communication on c.

Then we have for some p_2 :

$$p' \xrightarrow{c?} p_2 \text{ and } M = \textcircled{0} \parallel p_2 \ (\equiv p'')$$

Finally :

$$\begin{array}{ccccc} p & & \xrightarrow{\quad c! \quad} & & p' \\ \downarrow \tau & & & & \downarrow c? \\ p'' & & \equiv & & p_2 \end{array}$$

□

Chapter 4

Value Passing

4.1 Definition

In this chapter, we introduce the Value Passing. In the previous chapters, we talked about how to model communication. In addition, we will introduce value passing, to keep tracking the communication and the value (the message) that is sent or receive.

We have a countable set of variables called $\mathcal{V}$ and a countable set of constant, we take $\mathbb{N}$ for the rest of the document.

We can now introduce more specified Actions Prefixes for a process :

Definition 4.1.1 (ActionPrefix): We define the Action Prefixes as follow :

$$\pi ::= c\,?\,x \mid c\,!\,v \mid \tau$$
$$(c \in \mathcal{C}, x \in \mathcal{V}, v \in \mathcal{V} \cup \mathbb{N})$$

The Input prefix $c\,!\,x$ is the receiving of an information, that we call x, via the channel c.

The Output prefix $c\,!\,v$ is the sending of the information v via the channel c.

τ remains the same as for CCS or ACCS, it is the observable action.

Definition 4.1.2 (Free Variables and Bound Variables of Data): We say that the Action Prefix $c\,?\,x$ bounds the variable x in P for the process $c\,?\,x\,\bullet P$.
If a variable isn't bounded then it is free.

Example: The operation is the same as for lambda calculus : in $\lambda x.y\,x\,x$, x is bound by λx and y is free.

In $c\,?\,x\,\bullet(c\,!\,x\,\bullet\textcircled{0} \parallel c\,!\,y\,\bullet\textcircled{1})$, the variable x is bound and y is free.

Now, we take process up to the α-equivalence classes, and to avoid some comprehension problems, all bounded variables under an input action prefix are taken differently from each input action prefix, and from each free variables.

Example: For the process :

$$c\,?\,x\,\bullet P\,;\,c'\,?\,x\,\bullet P'$$

we take the representative :

$$c\,?\,x\,\bullet P\,;\,c'\,?\,y\,\bullet P'$$

Definition 4.1.3 (Substitution in Processes): The substitution of x in P by v (noted $M[x := v]$) is defined as for lambda calculus with bound and free variables.

Example: For the process :

$$(a\,!\,x \bullet ⓪ \parallel b\,!\,x \bullet ⓪)[x := 1] = (a\,!\,x \bullet ⓪)[x := 1] \parallel (b\,!\,1 \bullet ⓪)[x := 1]$$
$$= a\,!\,1 \bullet ⓪ \parallel b\,!\,1 \bullet ⓪$$

The Value that we are passing are destined for equation to make assertions in test.

We take $\mathcal{E}$ the set of our equations on $\mathcal{V} \cup \mathbb{N}$. The goal is to make an "If" constructor with condition in $\mathcal{E}$.

Example: We can make some sentences :

$$x = 4$$
$$mark > 10$$
$$age > 18 \wedge age \leqslant 30$$

With this set $\mathcal{E}$ comes a truth function, that can evaluate $Eq\ (\in \mathcal{E})$.

Example: Like in programming language :

$$\text{If } score = 4 \text{ Then } ① \text{ Else } ⓪$$

The implementation in COQ of the substituion is discussed in the appendix.

To summarize it, we used the De Brujin notation to simplify the α-equivalence.

4.2 CCS with Value passing (VCCS)

In this section we introduce CCS with Value passing (VCCS).

In comparison of CCS, we added the action prefixes, discussed in the previous section and a If . Then . Else . constructor to "use" the tranfered values.

Definition 4.2.1 (Processes, Guards and ActionPrefixes): The processes $\mathcal{P}_{\text{VCCS}}$ are :

(Process)
$$P, P_1, P_2 ::= P_1 \parallel P_2 \mid X \mid \text{rec}\, X.P \mid \text{If } Eq \text{ Then } P \text{ Else } Q \mid G$$

(Guards)
$$G, G_1, G_2 ::= ⓪ \mid ① \mid \pi \bullet P \mid G_1 + G_2$$

(ActionPrefix)
$$\pi ::= c\,?\,x \mid c\,!\,v \mid \tau$$
$$(c \in \mathcal{C}, x \in \mathcal{V}, v \in \mathcal{V} \cup \mathbb{N}, Eq \in \mathcal{E})$$

Just a reminder for the bound and free variables :

Example: In the process :

$$c\,?\,x \bullet (\text{If } x = 64 \text{ Then } a\,!\,① \bullet ⓪ \text{ Else } b\,!\,① \bullet ⓪) \parallel c\,?\,x \bullet ⓪$$

The first Input Prefix $c\,?\,x$ bounds x in :

$$\text{If } x = 64 \text{ Then } a\,!\,① \bullet ⓪ \text{ Else } b\,!\,① \bullet ⓪$$

And the Input Prefix $c\,?\,x$ bounds x in ⓪.
Then we take the repressentative :

$$c\,?\,x \bullet (\text{If } x = 64 \text{ Then } a\,!\,① \bullet ⓪ \text{ Else } b\,!\,① \bullet ⓪) \parallel c\,?\,y \bullet ⓪$$

Compared to CCS, we had :

- the constructor If . Then . Else .

- the new action prefixes

DEFINITION 4.2.2 (Context for $\mathcal{P}_{\text{VCCS}}$): A Context is a Process, following the definition 4.2.1, with one or multiple hole(s) (noted $(\!|\,|\!)$).

EXAMPLE: Let's take :

$$C := \text{If } (x = 17) \text{ Then } (\!|\,|\!) \text{ Else } (c\,?\,x \bullet \textcircled{0})$$

Then :

$$C(\!|\textcircled{1}|\!) := \text{If } (x = 17) \text{ Then } \textcircled{1} \text{ Else } (c\,?\,x \bullet \textcircled{0})$$

DEFINITION 4.2.3 (Structural Congruence for $\mathcal{P}_{\text{VCCS}}$): The Structural Congruence over $\mathcal{P}_{\text{VCCS}}$ is the reflexive, symmetric and transitive closure of the binary relation induced by the rules in figure 4.1.

$$\begin{array}{rll}
\text{[Sc-sneut]} & G + \textcircled{0} \equiv G & \\
\text{[Sc-scom]} & G_1 + G_2 \equiv G_2 + G_1 & \\
\text{[Sc-sass]} & G_1 + (G_2 + G_3) \equiv (G_1 + G_2) + G_3 & \\
\\
\text{[Sc-pneut]} & P \parallel \textcircled{0} \equiv P & \\
\text{[Sc-pcom]} & P \parallel Q \equiv Q \parallel P & \\
\text{[Sc-pass]} & P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R & \\
\\
\text{[Sc-context]} & C(\!|P|\!) \equiv C(\!|Q|\!) & if\, P \equiv Q
\end{array}$$

Figure 4.1: Structural Congruence rules for VCCS

DEFINITION 4.2.4 (Reduction Semantic for $\mathcal{P}_{\text{VCCS}}$): The Reduction Semantic is the binary relation over $\mathcal{P}_{\text{VCCS}}$ is defined in figure 4.2. It defines the STS $(\mathcal{P}_{\text{VCCS}}, \longrightarrow)$.

$$\text{[Sts-com]}\ \frac{}{c\,!\,v \bullet P_1 + G_1 \parallel c\,?\,x \bullet P_2 + G_2 \longrightarrow P_1 \parallel (P_2[x := v])} \qquad \text{[Sts-tau]}\ \frac{}{\tau \bullet P + G \longrightarrow P}$$

$$\text{[Sts-unf]}\ \frac{}{\text{rec } X.C(\!|X|\!) \longrightarrow C(\!|\text{rec } X.C(\!|X|\!)|\!)} \qquad \text{[Sts-par]}\ \frac{P_1 \longrightarrow P_2}{P_1 \parallel Q \longrightarrow P_2 \parallel Q}$$

$$\text{[Sts-if-True]}\ \frac{\phi(Eq) = True}{\text{If } Eq \text{ Then } P \text{ Else } Q \longrightarrow P} \qquad \text{[Sts-if-False]}\ \frac{\phi(Eq) = False}{\text{If } Eq \text{ Then } P \text{ Else } Q \longrightarrow Q}$$

$$\text{[Sts-cong]}\ \frac{P_1 \equiv Q_1 \quad Q_1 \longrightarrow Q_2 \quad Q_2 \equiv P_2}{P_1 \longrightarrow P_2}$$

Figure 4.2: The STS of $\mathcal{P}_{\text{VCCS}}$
The meta-variables are $c \in \mathcal{C}, x \in \mathcal{V}, v \in \mathcal{V} \cup \mathbb{N}$.

There are 3 new things in the STS :

- the reduction of If Eq Then . Else . if Eq is $True$
- the reduction of If Eq Then . Else . if Eq is $False$
- the communication

The reduction of If Eq Then . Else . is quite classic.

If the truth function ϕ evaluates the Equation Eq at $True$ then we reduce the term to its first branch and if the truth function ϕ evaluates the Equation Eq at $True$ then we reduce the term to its first branch.

Example: Take the process $Monday$:

$$\text{If } t > 20 \text{ Then } (\text{rec } X.Market\,!\,IceCream \bullet ⓪ \parallel X) \text{ Else } (\text{rec } X.Market\,!\,Coffee \bullet ⓪ \parallel X)$$

If I set the temperature t to 27, then Eq becomes $27 > 20$, Monday will reduce to :

$$Monday[t := 27] \longrightarrow \text{rec } X.(Market\,!\,IceCream \bullet ⓪ \parallel X)$$

We call this term $IceCreamDay$.

The rule [Sts-com], compared to CCS, now transfers the value from the output prefix and send it to the sub process of the input.

We can now continue our previous example.

Example: Take the processes :

$$\begin{aligned} &thermometer\,?\,t \bullet Monday \\ &V := thermometer\,!\,27 \bullet ⓪ \\ &Clients := \text{rec } X.(Market\,?\,y \bullet ① \parallel \tau \bullet X) \end{aligned}$$

Then I put then all in parallel :

$$thermometer\,?\,t \bullet Monday \parallel thermometer\,!\,27 \bullet ⓪ \parallel Clients$$

Let's reduce this term.

I open the shop on Monday. Let's look at the temperature, to see what I will sell. Then makes some food, and wait for some clients to come.

$$\begin{aligned} &(thermometer\,?\,t \bullet Monday) \parallel (thermometer\,!\,27 \bullet ⓪) \parallel Clients \\ &\longrightarrow Monday[t := 27] \parallel ⓪ \parallel Clients \\ &\longrightarrow IceCreamDay \parallel Clients \\ &\longrightarrow (Market\,!\,IceCream \bullet ⓪) \parallel IceCreamDay \parallel Clients \\ &\longrightarrow (Market\,!\,IceCream \bullet ⓪) \parallel IceCreamDay \parallel Market\,?\,y \bullet ① \parallel \tau \bullet Clients \\ &\longrightarrow ⓪ \parallel IceCreamDay \parallel ① \parallel \tau \bullet Clients \\ &\longrightarrow IceCreamDay \parallel Clients \parallel ① \end{aligned}$$

For the LTS, we have to redefine the set of Actions.

Definition 4.2.5 (Actions): We define the Actions $\mathcal{A}$ with :

- (the input-action) $c\,?\,z$

- (the output-action) $c\,!\,z$
- (the tau-action) τ

$(c \in \mathcal{C}, z \in \mathcal{V} \cup \mathbb{N})$

We added the description of which data the process input ou output.

Definition 4.2.6 (LTS for $\mathcal{P}_{\text{VCCS}}$): The LTS $(\mathcal{P}_{\text{VCCS}}, \mathcal{A}, \xrightarrow{\alpha})$ is defined in figure 4.3.

[Input] $\dfrac{}{c\,?\,x \bullet P \xrightarrow{c\,?\,v} P[x := v]}$ [Tau] $\dfrac{}{\tau \bullet P \xrightarrow{\tau} P}$

[Output] $\dfrac{}{c\,!\,v \bullet P \xrightarrow{c\,!\,v} P}$ [Unf] $\dfrac{}{\text{rec } X.C(\!|X|\!) \xrightarrow{\tau} C(\!|\text{rec } X.P|\!)}$

[Sum-L] $\dfrac{G \xrightarrow{\alpha} P}{G + G' \xrightarrow{\alpha} P}$ [Sum-R] $\dfrac{G \xrightarrow{\alpha} P}{G' + G \xrightarrow{\alpha} P}$

[Par-L] $\dfrac{P \xrightarrow{\alpha} Q}{P \parallel P' \xrightarrow{\alpha} Q \parallel P'}$ [Par-R] $\dfrac{P \xrightarrow{\alpha} Q}{P \parallel P' \xrightarrow{\alpha} Q \parallel P'}$

[Com-L] $\dfrac{P \xrightarrow{c\,?\,v} Q \quad P' \xrightarrow{c\,!\,v} Q'}{P \parallel Q \xrightarrow{\tau} Q \parallel Q'}$ [Com-R] $\dfrac{P \xrightarrow{c\,?\,v} Q \quad P' \xrightarrow{c\,!\,v} Q'}{P' \parallel P \xrightarrow{\tau} Q' \parallel Q}$

[If-True] $\dfrac{\phi(Eq) = True}{\text{If } Eq \text{ Then } P \text{ Else } Q \xrightarrow{\tau} P}$ [If-False] $\dfrac{\phi(Eq) = False}{\text{If } Eq \text{ Then } P \text{ Else } Q \xrightarrow{\tau} Q}$

Figure 4.3: The LTS of $\mathcal{P}_{\text{VCCS}}$
The meta-variables are $c \in \mathcal{C}, v \in \mathcal{V} \cup \mathbb{N}, \alpha \in \mathcal{A}$.

Like in the Reduction Semantic, the main additions are :

- value-passing for the [Input] rule
- reduction [If-True] and [If-False] for the If . Then . Else . constructor

$$\frac{}{c\,?\bullet P \xrightarrow{c\,?} P} \Rightarrow \frac{}{c\,?\,x \bullet P \xrightarrow{c\,?\,v} P[x := v]}$$

CCS $\Rightarrow$ VCCS

To maintain the properties that we had in CCS, we have introduce another lemma :

Lemma 4.2.7 (SubstitionRespectsCongruence): $\forall (P, Q) \in \mathcal{P}_{\text{VCCS}}, \forall x \in \mathcal{V}$ and $\forall v \in \mathcal{V} \cup \mathbb{N}$,

$$\text{if } P \equiv Q \text{ then } P[x := v] \equiv Q[x := v]$$

Proof. Let's take $(P, Q) \in \mathcal{P}_{\text{VCCS}}$, $x \in \mathcal{V}$ and $v \in \mathcal{V} \cup \mathbb{N}$.

Assume that we have :

$$P \equiv Q$$

We go by induction on

$$P \equiv Q$$

.

- [Sc-sneut]

  We have :

  $$G + ⓪ \equiv G$$

  By definition:

  $$(G + ⓪)[x := v] = G[x := v] + ⓪$$

  Then by [Sc-sneut]:

  $$G[x := v] + ⓪ \equiv G[x := v]$$

- [Sc-scom]

  We have :

  $$G_1 + G_2 \equiv G_2 + G_1$$

  By definition:

  $$(G_1 + G_2)[x := v] = G_1[x := v] + G_2[x := v]$$

  Then by [Sc-scom]:

  $$G_1[x := v] + G_2[x := v] \equiv G_2[x := v] + G_1[x := v]$$

  And by definition :

  $$G_2[x := v] + G_1[x := v] = (G_2 + G_1)[x := v]$$

- [Sc-sass]

  We have :

  $$G_1 + (G_2 + G_3) \equiv (G_1 + G_2) + G_3$$

  By definition:

  $$(G_1 + (G_2 + G_3))[x := v] = G_1[x := v] + (G_2[x := v] + G_3[x := v])$$

  Then by [Sc-sass]:

  $$G_1[x := v] + (G_2[x := v] + G_3[x := v]) \equiv (G_1[x := v] + G_2[x := v]) + G_3[x := v]$$

  And by definition :

  $$(G_1[x := v] + G_2[x := v]) + G_3[x := v] = ((G_1 + G_2) + G_3)[x := v]$$

- [Sc-pneut]

  We have :

  $$P \parallel ⓪ \equiv P$$

  By definition:

  $$(P \parallel ⓪)[x := v] = P[x := v] \parallel ⓪$$

  Then by [Sc-pneut]:

  $$P[x := v] \parallel ⓪ \equiv P[x := v]$$

- [Sc-pcom]

  We have :
  $$P \parallel Q \equiv Q \parallel P$$

  By definition:
  $$(P \parallel Q)[x := v] = P[x := v] \parallel Q[x := v]$$

  Then by [Sc-pcom]:
  $$P[x := v] \parallel Q[x := v] \equiv Q[x := v] \parallel P[x := v]$$

  And by definition :
  $$Q[x := v] \parallel P[x := v] = (Q \parallel P)[x := v]$$

- [Sc-pass]

  We have :
  $$PrlP(Q \parallel R) \equiv (P \parallel Q) \parallel R$$

  By definition:
  $$(P \parallel (Q \parallel R))[x := v] = P[x := v] \parallel (Q[x := v] \parallel R[x := v])$$

  Then by [Sc-pass]:
  $$P[x := v] \parallel (Q[x := v] \parallel R[x := v]) \equiv (P[x := v] \parallel Q[x := v]) \parallel R[x := v]$$

  And by definition :
  $$(P[x := v] \parallel Q[x := v]) \parallel R[x := v] = ((P \parallel Q) \parallel R)[x := v]$$

- [Sc-context]

  We have :
  $$C(\!|P|\!) \equiv C(\!|Q|\!)$$

  Obtained by :
  $$P \equiv Q$$

  By definition :
  $$(C(\!|P|\!))[x := v] = C[x := v](\!|P[x := v]|\!)$$

  And by induction hypothesis we have :
  $$P[x := v] \equiv Q[x := v]$$

  Then by [Sc-context] :
  $$C[x := v](\!|P[x := v]|\!) \equiv C[x := v](\!|Q[x := v]|\!)$$

  And by definition :
  $$C[x := v](\!|Q[x := v]|\!) = C(\!|Q|\!)[x := v]$$

□

Then we have our properties from CCS :

LEMMA 4.2.8 (ReductionShape): $\forall (P,Q) \in \mathcal{P}^2_{\text{VCCS}}$, if $P \longrightarrow Q$ then one of this cases occurs :

- $P \equiv c\,?\,x \bullet P_1 + G_1 \parallel c\,!\,v \bullet P_2 + G_2 \parallel S$ and $Q \equiv P_1 \parallel P_2[x := v] \parallel S$ for some $P_1, G_1, P_2, G_2, S, x, v$
- $P \equiv \tau \bullet P_1 + G_1 \parallel S$ and $Q \equiv P_1 \parallel S$ for some P_1, G_1, S
- $P \equiv \text{rec}\, X.C(\!|X|\!) \parallel S$ and $Q \equiv C(\!|\text{rec}\, X.C(\!|X|\!)|\!) \parallel S$ for some C, S
- $P \equiv \text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \parallel S, Q \equiv P_1 \parallel S$ and $\phi(Eq) = True$ for some Eq, P_1, P_2, S
- $P \equiv \text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \parallel S, Q \equiv P_2 \parallel S$ and $\phi(Eq) = False$ for some Eq, P_1, P_2, S

Proof. For all constructors from CCS, see the proof of lemma 2.2.4.

Otherwise, it is the same proof for the new constructors :

- For [STS-COM], we have :

$$\frac{}{c\,!\,v \bullet P_1 + G_1 \parallel c\,?\,x \bullet P_2 + G_2 \longrightarrow P_1 \parallel (P_2[x := v])}$$

  We have by [SC-PNEUT]:

$$c\,!\,v \bullet P_1 + G_1 \parallel c\,?\,x \bullet P_2 + G_2 \equiv c\,!\,v \bullet P_1 + G_1 \parallel c\,?\,x \bullet P_2 + G_2 \parallel ⓪$$

  And :

$$P_1 \parallel (P_2[x := v]) \equiv P_1 \parallel (P_2[x := v]) \parallel ⓪$$

- For [STS-IF-TRUE], we have :

$$\frac{\phi(Eq) = True}{\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \longrightarrow P_1}$$

  We have by [SC-PNEUT]:

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \equiv \text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \parallel ⓪$$

  And :

$$P_1 \equiv P_1 \parallel ⓪$$

- For [STS-IF-FALSE], we have :

$$\frac{\phi(Eq) = False}{\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \longrightarrow P_2}$$

  We have by [SC-PNEUT]:

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \equiv \text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \parallel ⓪$$

  And :

$$P_2 \equiv P_2 \parallel ⓪$$

□

LEMMA 4.2.9 (SimplifiedTree): $\forall (P,Q) \in \mathcal{P}_{\text{VCCS}}$, if $P \longrightarrow Q$, then there exists a proof tree of length 3 that deduce $P \longrightarrow Q$.

LEMMA 4.2.10 (TransitionShapeForInput): $\forall (P, Q) \in \mathcal{P}_{\text{VCCS}}$, $c \in \mathcal{C}$ and $v \in \mathcal{V} \cup \mathbb{N}$, if $P \xrightarrow{c\,?\,v} Q$, then $P \equiv (c\,?\,x \bullet P_1 + G_1) \parallel S$, $Q \equiv P_1[x := v] \parallel S$ for some P_1, G_1, S.

If P is a guard then $S = \textcircled{0}$.

Proof. For all constructors from CCS, see the proof of lemma 2.3.3.

Otherwise, it is the same proof for the new constructor :

- For [INPUT]

$$\frac{}{c\,?\,x \bullet P \xrightarrow{c\,?\,v} P[x := v]}$$

  By [SC-SNEUT] and [SC-PNEUT], we have :

$$c\,?\,x \bullet P \equiv c\,?\,x \bullet P + \textcircled{0} \parallel \textcircled{0}$$

□

LEMMA 4.2.11 (TransitionShapeForOutput): $\forall (P, Q) \in \mathcal{P}_{\text{VCCS}}$, $c \in \mathcal{C}$ and $v \in \mathcal{V} \cup \mathbb{N}$, if $P \xrightarrow{c\,!\,v} Q$, then $P \equiv (c\,!\,x \bullet P_1 + G_1) \parallel S$, $Q \equiv P_1 \parallel S$ for some P_1, G_1, S.

If P is a guard then $S = \textcircled{0}$.

Proof. See the proof of lemma 2.2.4. □

LEMMA 4.2.12 (TransitionShapeForTauAndGuard): $\forall (P, Q) \in \mathcal{P}_{\text{VCCS}}$ and $c \in \mathcal{C}$, if P is a guard and $P \xrightarrow{\tau} Q$, then $P \equiv \tau \bullet P_1 + G_1$, $Q \equiv P_1$ for some P_1, G_1.

Proof. The constructor If . Then . Else . isn't a guard.

Then, see the proof of lemma 2.3.5. □

LEMMA 4.2.13 (CongruenceRespectsTransition): $\forall (S, T) \in \mathcal{P}^2_{\text{VCCS}}$, $\alpha \in \mathcal{A}$, if $S \equiv . \xrightarrow{\alpha} T$ then $S \xrightarrow{\alpha} . \equiv T$.

Proof. We follow the proof of lemma 2.4.1.

The only case that is different from lemma 2.4.1 is when the context $C = c\,?\,v \bullet ()$ for the rule [SC-CONTEXT].

We need one step further, we have it by the lemma 4.2.7 :

Assume that we have :

$$c\,?\,x \bullet P_1 \equiv c\,?\,x \bullet P_2 \xrightarrow{c\,?\,v} P_2[x := v]$$

by $P_1 \equiv P_2$.

Then by [INPUT], we have :

$$c\,?\,x \bullet P_1 \xrightarrow{c\,?\,v} P_1[x := v]$$

and by lemma 4.2.7, we have :

$$P_1[x := v] \equiv P_2[x := v]$$

We can talk a bit about the context $C = \text{If } Eq \text{ Then } () \text{ Else } P_2$ or $C = \text{If } Eq \text{ Then } P_1 \text{ Else } ()$, since the the Structural Congruence doesn't modify the equation Eq, we have :

- Assume that we have :

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \equiv \text{If } Eq \text{ Then } P_1 \text{ Else } P_2' \xrightarrow{\tau} P_1$$

by $P_2 \equiv P_2'$ and $\phi(Eq) = True$.

Then by [If-True] and reflexivity, we have :

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \xrightarrow{\tau} P_1 \equiv P_1$$

- Assume that we have :

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \equiv \text{If } Eq \text{ Then } P_1 \text{ Else } P_2' \xrightarrow{\tau} P_2'$$

by $P_2 \equiv P_2'$ and $\phi(Eq) = False$.

Then by [If-True] and hypothesis, we have :

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \xrightarrow{\tau} P_2 \equiv P_2'$$

- Assume that we have :

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \equiv \text{If } Eq \text{ Then } P_1' \text{ Else } P_2 \xrightarrow{\tau} P_1'$$

by $P_1 \equiv P_1'$ and $\phi(Eq) = True$.

Then by [If-True] and hypothesis, we have :

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \xrightarrow{\tau} P_1 \equiv P_1'$$

- Assume that we have :

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \equiv \text{If } Eq \text{ Then } P_1' \text{ Else } P_2 \xrightarrow{\tau} P_2$$

by $P_1 \equiv P_1'$ and $\phi(Eq) = False$.

Then by [If-True] and reflexivity, we have :

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \xrightarrow{\tau} P_2 \equiv P_2'$$

□

Lemma 4.2.14 (Harmony Lemma): For all $(S, T) \in \mathcal{P}^2_{\text{CCS}}$, $S \longrightarrow T$ if and only if $S \xrightarrow{\tau} . \equiv T$.

Proof. We have two side to prove :

- if $S \longrightarrow T$ then $S \xrightarrow{\tau} . \equiv T$
- if $S \xrightarrow{\tau} . \equiv T$ then $S \longrightarrow T$

In each side, we follow the same schema as for the lemma 2.4.2, but let's precise the proof for the new constructors, reductions and transitions.

- For the first side, we remember that we used lemma 4.2.8, to reason by case analysis.

  So :

– We have for some Eq, P_1, P_2, S:

$$P \equiv (\text{If } Eq \text{ Then } P_1 \text{ Else } P_2) \parallel S$$
$$Q \equiv P_1 \parallel S$$

by $\phi(Eq) = True$.

Then by [If-True] and [Par-L], we have :

$$(\text{If } Eq \text{ Then } P_1 \text{ Else } P_2) \parallel S \xrightarrow{\tau} P_1 \parallel S$$

And by lemma 4.2.13, it exists M such that :

$$P \xrightarrow{\tau} M \equiv P_1 \parallel S \, (\equiv Q)$$

– We have for some Eq, P_1, P_2, S:

$$P \equiv (\text{If } Eq \text{ Then } P_1 \text{ Else } P_2) \parallel S$$
$$Q \equiv P_2 \parallel S$$

by $\phi(Eq) = False$.

Then by [If-False] and [Par-L], we have :

$$(\text{If } Eq \text{ Then } P_1 \text{ Else } P_2) \parallel S \xrightarrow{\tau} P_2 \parallel S$$

And by lemma 4.2.13, it exists M' such that :

$$P \xrightarrow{\tau} M' \equiv P_2 \parallel S \, (\equiv Q)$$

– We have for some $P_1, P_2, G_1, G_2, x, v, S$:

$$P \equiv c\,?\,x \bullet P_1 + G_1 \parallel c\,!\,v \bullet P_2 + G_2 \parallel S$$
$$Q \equiv P_1[x := v] \parallel P_2 \parallel S$$

Then by [Sum-L], [Com-L] and [Par-L], we have :

$$c\,?\,x \bullet P_1 + G_1 \parallel c\,!\,v \bullet P_2 + G_2 \parallel S \xrightarrow{\tau} P_1[x := v] \parallel P_2 \parallel S$$

And by lemma 4.2.13, it exists M" such that :

$$P \xrightarrow{\tau} M'' \equiv P_1[x := v] \parallel P_2 \parallel S \, (\equiv Q)$$

■ For the second side,

– We have for some Eq, P_1, P_2:

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \xrightarrow{ActTau} P_1$$

for $\phi(Eq) = True$.

By [Sts-if-True] and reflexivity, we have :

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \longrightarrow P_1 \equiv P_1$$

– We have for some Eq, P_1, P_2:

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \xrightarrow{ActTau} P_1$$

for $\phi(Eq) = False$.

By [Sts-if-True] and reflexivity, we have :

$$\text{If } Eq \text{ Then } P_1 \text{ Else } P_2 \longrightarrow P_1 \equiv P_1$$

– We have for some $c, x, v, P_1, P_2, P_1', P_2$:

$$P_1 \parallel P_2 \xrightarrow{\tau} P_1' \parallel P_2'$$

by :

$$P_1 \xrightarrow{c\,?\,v} P_1'$$
$$P_2 \xrightarrow{c\,!\,v} P_2'$$

By lemma 4.2.10 and lemma 4.2.11, we have for some G_1, G_2, S_1, S_2 :

$$P_1 \equiv c\,?\,x \bullet Q_1 + G_1 \parallel S_1$$
$$P_1' \equiv Q_1[x := v] \parallel S_1$$

$$P_2 \equiv c\,!\,x \bullet Q_2 + G_2 \parallel S_2$$
$$P_2' \equiv Q_2 \parallel S_2$$

By [Sts-com] and [Sts-par] with $S_1 \parallel S_2$, we have :

$$c\,?\,x \bullet Q_1 + G_1 \parallel c\,!\,x \bullet Q_2 + G_2 \parallel S_1 \parallel S_2 \longrightarrow Q_1[x := v] \parallel Q_2 \parallel S_1 \parallel S_2$$

But we have :

$$P_1 \parallel P_2 \equiv c\,?\,x \bullet Q_1 + G_1 \parallel c\,!\,x \bullet Q_2 + G_2 \parallel S_1 \parallel S_2$$
$$P_1' \parallel P_2' \equiv Q_1[x := v] \parallel Q_2 \parallel S_1 \parallel S_2$$

Then by [Sts-cong], we have :

$$P_1 \parallel P_2 \longrightarrow P_1' \parallel P_2'$$

□

4.3 ACCS with Value passing (VACCS)

In this chapter, we introduce VACCS.

The idea from CCS to ACCS remains the same from VCCS to VACCS :

- the output process $c\,!\,v \bullet P$ is no longer a guard

- in the output process $c\,!\,v\bullet P$, $P = \textcircled{0}$

DEFINITION 4.3.1 (Processes, Guards and ActionPrefixes): The processes $\mathcal{P}_{\text{VACCS}}$ are :

(Process)
$$P, P_1, P_2 ::= P_1 \parallel P_2 \mid X \mid \text{rec}\, X.P \mid \text{If}\, C\, \text{Then}\, P\, \text{Else}\, Q \mid G$$

(Guards)
$$G, G_1, G_2 ::= \textcircled{0} \mid \textcircled{1} \mid \pi \bullet P \mid G_1 + G_2$$

(ActionPrefix)
$$\pi ::= c\,?\,x \mid c\,!\,v \mid \tau$$

($c \in \mathcal{C}, x \in \mathcal{V}, v \in \mathcal{V} \cup \mathbb{N}$)

DEFINITION 4.3.2 (Context for $\mathcal{P}_{\text{VACCS}}$): A Context is a Process, following the definition 4.3.1, with one or multiple hole(s) (noted $(\!|\,|\!)$).
We note $C(\!|P|\!)$, where the hole(s) in C are substituted by P.
Note, that a hole can be in place for guards (example in the summation), then it can only be replaced by a guard G.

EXAMPLE: Let's take $C :=$ If $(x = 17)$ Then $(\!|\,|\!)$ Else $(c\,?\,x \bullet \textcircled{0})$.
Then $C(\!|\textcircled{1}|\!) :=$ If $(x = 17)$ Then $\textcircled{1}$ Else $(c\,?\,x \bullet \textcircled{0})$.

DEFINITION 4.3.3 (Structural Congruence for $\mathcal{P}_{\text{VACCS}}$): The Structural Congruence over $\mathcal{P}_{\text{VACCS}}$ is the reflexive, symetric and transitive closure of the binary relation induced by the rules in figure 4.4.

$$\begin{aligned}
&[\text{Sc-sneut}]\ G + \textcircled{0} \equiv G \\
&[\text{Sc-scom}]\ G_1 + G_2 \equiv G_2 + G_1 \\
&[\text{Sc-sass}]\ G_1 + (G_2 + G_3) \equiv (G_1 + G_2) + G_3 \\
&\\
&[\text{Sc-pneut}]\ P \parallel \textcircled{0} \equiv P \\
&[\text{Sc-pcom}]\ P \parallel Q \equiv Q \parallel P \\
&[\text{Sc-pass}]\ P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \\
&\\
&[\text{Sc-context}]\ C(\!|P|\!) \equiv C(\!|Q|\!) \qquad \text{if } P \equiv Q
\end{aligned}$$

Figure 4.4: Structural congruence rules for VACCS

DEFINITION 4.3.4 (STS for $\mathcal{P}_{\text{VACCS}}$): The STS $(\mathcal{P}_{\text{VACCS}}, \longrightarrow)$ is defined in figure 4.5.

DEFINITION 4.3.5 (LTS for $\mathcal{P}_{\text{VACCS}}$): The LTS $(\mathcal{P}_{\text{VACCS}}, \mathcal{A}, \xrightarrow{\alpha})$ is defined in figure 4.6.

LEMMA 4.3.6 (ReductionShape): $\forall (P, Q) \in \mathcal{P}^2_{\text{VACCS}}$, if $P \longrightarrow Q$ then one of this cases occurs :

- $P \equiv c\,!\,v \bullet \textcircled{0} \parallel c\,?\,x \bullet P_2 + G_2 \parallel S$ and $Q \equiv P_1 \parallel S$ for some P_2, G_2, S, x, v
- $P \equiv \tau \bullet P_1 + G_1 \parallel S$ and $Q \equiv P_1 \parallel S$ for some P_1, G_1, S
- $P \equiv \text{rec}\, X.C(\!|X|\!) \parallel S$ and $Q \equiv C(\!|\text{rec}\, X.C(\!|X|\!)|\!) \parallel S$ for some C, S
- $P \equiv \text{If}\, Eq\, \text{Then}\, P_1\, \text{Else}\, P_2 \parallel S$, $Q \equiv P_1 \parallel S$ and $\phi(Eq) = True$ for some Eq, P_1, P_2, S
- $P \equiv \text{If}\, Eq\, \text{Then}\, P_1\, \text{Else}\, P_2 \parallel S$, $Q \equiv P_2 \parallel S$ and $\phi(Eq) = False$ for some Eq, P_1, P_2, S

Proof. See the proof of lemma 3.2.1 and for the substitution cases, the proof of lemma 4.2.8. □

[STS-COM] $\overline{c\,!\,v \bullet ⓪ \parallel c\,?\,x \bullet P_2 + G_2 \longrightarrow P_1 \parallel (P_2[x := v])}$ [STS-TAU] $\overline{\tau \bullet P + G \longrightarrow P}$

[STS-UNF] $\overline{\text{rec}\, X.C(\!|X|\!) \longrightarrow C(\!|\text{rec}\, X.C(\!|X|\!)|\!)}$ [STS-PAR] $\dfrac{P_1 \longrightarrow P_2}{P_1 \parallel Q \longrightarrow P_2 \parallel Q}$

[STS-IF-TRUE] $\dfrac{\phi(Eq) = True}{\text{If } Eq \text{ Then } P \text{ Else } Q \longrightarrow P}$ [STS-IF-FALSE] $\dfrac{\phi(Eq) = False}{\text{If } Eq \text{ Then } P \text{ Else } Q \longrightarrow Q}$

[STS-CONG] $\dfrac{P_1 \equiv Q_1 \quad Q_1 \longrightarrow Q_2 \quad Q_2 \equiv P_2}{P_1 \longrightarrow P_2}$

Figure 4.5: The STS of $\mathcal{P}_{\text{VACCS}}$
The meta-variables are $c \in \mathcal{C}, x \in \mathcal{V}, v \in \mathcal{V} \cup \mathbb{N}$.

LEMMA 4.3.7 (SubstitionRespectsCongruence): $\forall (P, Q) \in \mathcal{P}_{\text{VACCS}}$, $\forall x \in \mathcal{V}$ and $\forall v \in \mathcal{V} \cup \mathbb{N}$,

$$\text{if } P \equiv Q \text{ then } P[x := v] \equiv Q[x := v]$$

Proof. See the proof of lemma 4.2.7 □

LEMMA 4.3.8 (TransitionShapeForInput): $\forall (P, Q) \in \mathcal{P}_{\text{VACCS}}$, $c \in \mathcal{C}$ and $v \in \mathcal{V} \cup \mathbb{N}$, if $P \xrightarrow{c\,?\,v} Q$, then $P \equiv (c\,?\,x \bullet P_1 + G_1) \parallel S$, $Q \equiv P_1[x := v] \parallel S$ for some P_1, G_1, S.

If P is a guard then $S = ⓪$.

Proof. See the proof of lemma 2.3.3 and for the substitution cases, the proof of lemma 4.2.10. □

LEMMA 4.3.9 (TransitionShapeForOutput): $\forall (P, Q) \in \mathcal{P}_{\text{VACCS}}$, $c \in \mathcal{C}$ and $v \in \mathcal{V} \cup \mathbb{N}$, if $P \xrightarrow{c\,!\,v} Q$, then $P \equiv (c\,!\,x \bullet ⓪) \parallel S$, $Q \equiv S$ for some S.

Proof. See the proof of lemma 4.2.10. □

LEMMA 4.3.10 (TransitionShapeForTauAndGuard): $\forall (P, Q) \in \mathcal{P}_{\text{VACCS}}$ and $c \in \mathcal{C}$, if P is a guard and $P \xrightarrow{\tau} Q$, then $P \equiv \tau \bullet P_1 + G_1$, $Q \equiv P_1$ for some P_1, G_1.

Proof. See the proof of lemma 4.2.12 and lemma 3.2.5. □

LEMMA 4.3.11 (CongruenceRespectsTransition): $\forall (S, T) \in \mathcal{P}^2_{\text{VACCS}}$, $\alpha \in \mathcal{A}$, if $S \equiv . \xrightarrow{\alpha} T$ then $S \xrightarrow{\alpha} . \equiv T$.

Proof. See proof of lemma 4.2.13 and lemma 3.2.6. □

LEMMA 4.3.12 (Harmony Lemma): For all $(S, T) \in \mathcal{P}^2_{\text{VACCS}}$, $S \longrightarrow T$ if and only if $S \xrightarrow{\tau} . \equiv T$.

Proof. See proof of lemma 4.2.14 and lemma 3.2.7. □

Now we have to adapt the First-order axioms for out-buffered agents with feedback from Peter Selinger for VACCS.

By out x, it is designed for "an output action that we call x".

Then out x becomes $c\,!\,v$ for some c and v.

And out $x =$ out y means for VACCS , $c\,!\,v = c'\,!\,v'$.

And by construction, $c = c'$ and $v = v'$.

$$[\textsc{Input}]\ \frac{}{c?\,x \bullet P \xrightarrow{c?\,v} P[x := v]} \qquad [\textsc{Tau}]\ \frac{}{\tau \bullet P \xrightarrow{\tau} P}$$

$$[\textsc{Output}]\ \frac{}{c!\,v \bullet \textcircled{0} \xrightarrow{c!\,v} \textcircled{0}} \qquad [\textsc{Unf}]\ \frac{}{\text{rec}\,X.C(\!|X|\!) \xrightarrow{\tau} C(\!|\text{rec}\,X.P|\!)}$$

$$[\textsc{Sum-L}]\ \frac{G \xrightarrow{\alpha} P}{G + G' \xrightarrow{\alpha} P} \qquad [\textsc{Sum-R}]\ \frac{G \xrightarrow{\alpha} P}{G' + G \xrightarrow{\alpha} P}$$

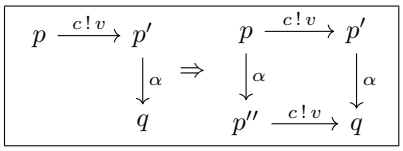
$$[\textsc{Par-L}]\ \frac{P \xrightarrow{\alpha} Q}{P \parallel P' \xrightarrow{\alpha} Q \parallel P'} \qquad [\textsc{Par-R}]\ \frac{P \xrightarrow{\alpha} Q}{P \parallel P' \xrightarrow{\alpha} Q \parallel P'}$$

$$[\textsc{Com-L}]\ \frac{P \xrightarrow{c?\,v} Q \quad P' \xrightarrow{c!\,v} Q'}{P \parallel Q \xrightarrow{\tau} Q \parallel Q'} \qquad [\textsc{Com-R}]\ \frac{P \xrightarrow{c?\,v} Q \quad P' \xrightarrow{c!\,v} Q'}{P' \parallel P \xrightarrow{\tau} Q' \parallel Q}$$

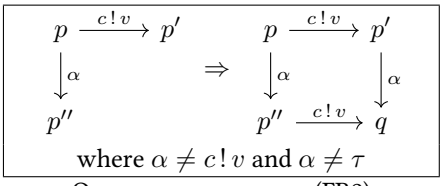
$$[\textsc{If-True}]\ \frac{\phi(Eq) = True}{\text{If } Eq \text{ Then } P \text{ Else } Q \xrightarrow{\tau} P} \qquad [\textsc{If-False}]\ \frac{\phi(Eq) = False}{\text{If } Eq \text{ Then } P \text{ Else } Q \xrightarrow{\tau} Q}$$

Figure 4.6: The LTS of $\mathcal{P}_{\text{VACCS}}$
The meta-variables are $c \in \mathcal{C}, v \in \mathcal{V} \cup \mathbb{N}, \alpha \in \mathcal{A}$.

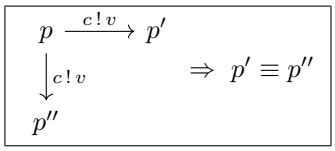
Finally we have :



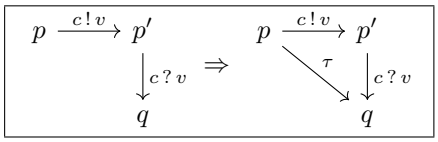
Output-commutativity (FB1)



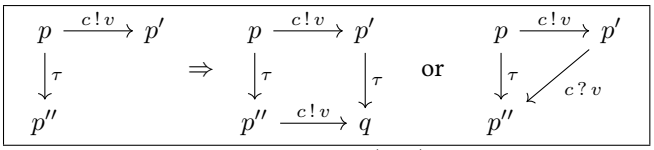
Output-confluence (FB2)



Output-determinacy (FB3)



Feedback (FB4)



Output-tau (FB5)

The proofs follow exactly the same schema as the proofs in ACCS.

Output-commutativity (FB1). See the proof in section 3.3. □

Output-confluence (FB2). See the proof in section 3.3. □

Output-determinacy (FB3). See the proof in section 3.3. □

Feedback (FB4). See the proof in section 3.3. □

Output-tau (FB5). See the proof in section 3.3. □

Chapter 5

CCS and Sequencing

In this chapter, we added the notion of Program Sequencing and we modify a bit the aspect of the processes.

Our objective is to build a asynchronous programming language with enough expressiveness.

The 2 things we have to implement are :

- the notion of sequencing
- the notion of skipping to the next instruction for ⓪

And then verify that the Harmony Lemma and the Axioms of Output Buffered agents are still good.

5.1 Definition

We have already a notion of sequencing, indeed with the action prefix :

$$c\,?\,x \bullet\tau \bullet\tau \bullet\tau \bullet c\,!\,v \bullet ⓪$$

But the τ is only here to say "you have a computation".

It says nothing about how it is computed, with enough τ-action like for the recursion or the If . Then . Else . constructor we would like to suppress this Action Prefix.

A start was to add the "If" constructor with his τ transition.

But now, we would like a prefix to say more that output, input or "something" is computed.

We want a more explicit grammar than "something".

We would like something like :

$$P\,;\,Q$$

And then the example above becomes :

$$c\,?\,x \bullet P_1\,;\,P_2\,;\,P_3\,;\,P_4; c\,!\,v \bullet ⓪$$

The sequencing would function like a parallel, but only on the left side, similar to a [Par-L] rule.

But then if P is an output process, like $P = c\,!\,v \bullet ⓪$, it would block the action of Q in :

$$P\,;\,Q$$

and would be a counter example of OUTPUT-COMMUTATIVITY (FB1).

So a rule like [PAR-R] would be needed for an output process in the first part of a sequencing process and a computation like this would be possible :

$$c\,!\,v \bullet ⓪ ;\ I_2 \xrightarrow{\tau} c\,!\,v \bullet ⓪ ;\ I_3 \xrightarrow{\tau} c\,!\,v \bullet ⓪ ;\ I_4 \xrightarrow{\tau} ...$$

We would like to "discard" the instructions in our sequencing.

Then we introduce a notion of a MailBox in "parallel" of the Processes, that stores the outputs and distribute them when needed by an input.

We first introduce our definition of Processes.

DEFINITION 5.1.1 (Processes, Guards and ActionPrefixes): The processes $\mathcal{P}_{\text{SVCCS}}$ are :

(Process)
$$P, P_1, P_2 ::= P_1 \parallel P_2 \mid X \mid \text{rec}\, X.P \mid \text{If}\, C\, \text{Then}\, P_1\, \text{Else}\, P_2 \mid ⓪ \mid G$$

(Guards)
$$G, G_1, G_2 ::= \delta \mid ① \mid \pi \bullet P \mid G_1 + G_2 \mid P_1 \,;\, P_2$$

(ActionPrefix)
$$\pi ::= c\,?\,x \ \mid c\,!\,v \mid \tau$$

$$(x \in \mathcal{V}, v \in \mathcal{V} \cup \mathbb{N})$$

Compared to VCCS, we added a few things :

- the process ⓪ is no longer a guard
- the deadlock process δ
- the sequencing process $P\,;\,Q$

⓪ don't block the sequencing but δ blocks everything in sequence.

The actions prefixes remain the same at one exception point of view : the "output" process is no longer viewed as an output for process : you can't communicate though it, or not directly.
We will see more about this in the transition part for the States (Mailbox with Process).

DEFINITION 5.1.2 (Context): A Context is a Process, following the definition 5.1.1, with one or multiple hole(s) (noted $(\!|\,|\!)$).

EXAMPLE: Let's take $C = (\!|\,|\!) \parallel c\,?\,x \bullet \delta \parallel (\!|\,|\!)$.
Then $C(\!|⓪|\!) = ⓪ \parallel c\,?\,x \bullet \delta \parallel ⓪$.

DEFINITION 5.1.3 (Actions): Actions (noted $\mathcal{A}$) is defined as below :

(Actions)
$$\mathcal{A} ::= c\,?\,v \ \mid c\,!\,v \mid \tau$$

$(c \in \mathcal{C}, v \in \mathcal{V} \cup \mathbb{N})$

The action $c\,!\,v$ is now viewed as storing in a MailBox.

DEFINITION 5.1.4 (Structural Congruence for Process): The Structural Congruence over the Processes is the reflexive, symetric and transitive closure of the binary relation induced by the rules in figure 5.1.

$$
\begin{array}{rll}
\text{[Sc-sneut]} & G + \delta \equiv G & \\
\text{[Sc-scom]} & G_1 + G_2 \equiv G_2 + G_1 & \\
\text{[Sc-sass]} & G_1 + (G_2 + G_3) \equiv (G_1 + G_2) + G_3 & \\
\\
\text{[Sc-pcom]} & P \parallel Q \equiv Q \parallel P & \\
\text{[Sc-pass]} & P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R & \\
\\
\text{[Sc-context]} & C(\!|P|\!) \equiv C(\!|Q|\!) & if\, P \equiv Q \\
\\
\text{[Sc-sassoc]} & P\,;\,(Q\,;\,R) \equiv (P\,;\,Q)\,;\,R & \\
\text{[Sc-pi-assoc]} & (\pi \bullet P)\,;\,Q \equiv \pi \bullet (P\,;\,Q) \quad (\pi \in \{\tau\,,\, c\,!\,v\}) & \\
\text{[Sc-input-assoc]} & (c\,?\,x \bullet P)\,;\,Q \equiv c\,?\,x \bullet (P\,;\,Q) \quad (x \notin \mathtt{FV}(Q)) & \\
\\
\text{[Sc-ssdist]} & (G_1 + G_2)\,;\,Q \equiv (G_1\,;\,Q) + (G_2\,;\,Q) &
\end{array}
$$

Figure 5.1: Structural congruence rules

The first six rules are pretty similar to the previous VCCS but with a few differences :

We avoided the rules :

$$P \parallel ⓪ \equiv P$$

to keep the moral of a guard is equivalent to a guard.

⓪ is no longer a guard, so we added the δ in the rule [Sc-sneut] :

$$G + \delta \equiv G$$

to keep some nice properties of transition/reduction shape.

Let's talk about the new rules :

- The rule [Sc-sassoc] is the associativity of the sequencing :

  $$P\,;\,(Q\,;\,R) \equiv (P\,;\,Q)\,;\,R$$

  what is quite fair for the sequencing.

- The next two rules are here for considering that the action-prefixing are in sense like sequencing then the associativity rules remains, [Sc-piassoc] :

  $$(\pi \bullet P)\,;\,Q \equiv \pi \bullet (P\,;\,Q) \quad (\pi \in \{\tau\,,\, c\,!\,v\})$$

  Nothing special to say about those two prefixe : they don't capture free variable.

  But let's look a a bit more about [Sc-input-assoc] :

  $$(c\,?\,x \bullet P)\,;\,Q \equiv c\,?\,x \bullet (P\,;\,Q) \quad (x \notin \mathtt{FV}(Q))$$

  The process $c\,?\,x \bullet P$ has nothing to do with Q, then if we don't put a restriction like $(x \notin \mathtt{FV}(Q))$, this will change the fundamental role of the whole process.

Capturing the variable x in Q, and then by communication, rewriting on the x's in Q and will destroy our lemma for congruence as bisimulation (lemma 4.2.13).

If we consider the rule as a relation left to right, the relation is always "possible".

Take the process :

$$(c\,?\,x \bullet a\,!\,x \bullet ⓪)\,;\ \text{If } (x > 10) \text{ Then } ① \text{ Else } \delta$$

With the α-equivalence, we can take the representative :

$$(c\,?\,y \bullet a\,!\,y \bullet ⓪)\,;\ \text{If } (x > 10) \text{ Then } ① \text{ Else } \delta$$

We just have to take one variable that doesn't belong in $\texttt{FV}(Q)$ and this is always possible, like in the example. And :

$$(c\,?\,y \bullet a\,!\,y \bullet ⓪)\,;\ \text{If } (x > 10) \text{ Then } ① \text{ Else } \delta \equiv c\,?\,y \bullet (a\,!\,y \bullet ⓪;\ \text{If } (x > 10) \text{ Then } ① \text{ Else } \delta)$$

In the other hand, if we take this rule right to left, the relation isn't always possible, if the input captures a variable in Q.

Take the process :

$$c\,?\,x \bullet (a\,!\,y \bullet ⓪;\ \text{If } (x > 10) \text{ Then } ① \text{ Else } \delta)$$

Then we can't disassemble the process "If $(x > 10)$ Then ① Else δ" from the input prefix $c\,?\,x$.

Indeed, if we do that :

$$(c\,?\,x \bullet a\,!\,y \bullet ⓪)\,;\ \text{If } (x > 10) \text{ Then } ① \text{ Else } \delta$$

the variable x in "If$(x > 10)$Then①Elseδ" won't be able to be substituted, and we will lose the lemma 4.2.13.

- the last rule, [Sc-ssdist], is a rule from CSP([23]), that is needed to keep some nice properties (in a certain way), which follows in the document (like lemma 4.2.10).

Before talking about the LTS for Process, and right after, the States and Mailboxes, we introduce a predicate : $SKIP(.)$.

That define the processes that will pass to the next instruction in the Sequencing, the reasonnable definition is the process ⓪ or multiple ⓪ in parallel.

Like if we start multiple tasks in parallel (like Merge sort), then when all tasks are finished (⓪), we skip to the next step.

Definition 5.1.5 ($SKIP$): We said that a process P skips to the next instruction if is in the form

- ⓪
- ⓪ ∥ ⓪ ∥ ... ∥ ⓪

Definition 5.1.6 (LTS for Process): The LTS $(\mathcal{P}_{\text{SVCCS}}, \mathcal{A}, \xrightarrow{\alpha})$ is defined in figure 5.2.

There is no more communication. The communication disapears : it will be done later with the MailBox.

We just send the information to the Mailbox via $c\,!\,v$.

And there are the two new rules for the Sequencing :

$$\text{[SKIP]}\ \frac{SKIP(P)}{P\,;\,Q \xrightarrow{\tau} Q} \qquad \text{[Seq]}\ \frac{P_1 \xrightarrow{\alpha} P_2}{P_1\,;\,Q \xrightarrow{\alpha} P_2\,;\,Q}$$

As we talked before, the [SKIP] rule is here to pass to the next instruction.

$$[\textsc{Input}]\ \frac{}{c\,?\,x\bullet P \xrightarrow{c\,?\,v} P[x := v]} \qquad [\textsc{Tau}]\ \frac{}{\tau\bullet P \xrightarrow{\tau} P}$$

$$[\textsc{Output}]\ \frac{}{c\,!\,z\bullet P \xrightarrow{c\,!\,z} P} \qquad [\textsc{Unf}]\ \frac{}{\text{rec}\,X.C(\!|X|\!) \xrightarrow{\tau} C(\!|\text{rec}\,X.P|\!)}$$

$$[\textsc{Sum-L}]\ \frac{G_1 \xrightarrow{\alpha} P_1}{G_1 + G \xrightarrow{\alpha} P_1} \qquad [\textsc{Sum-R}]\ \frac{G_1 \xrightarrow{\alpha} P_1}{G + G_1 \xrightarrow{\alpha} P_1}$$

$$[\textsc{Par-L}]\ \frac{P_1 \xrightarrow{\alpha} P_2}{P_1 \parallel P \xrightarrow{\alpha} P_2 \parallel P} \qquad [\textsc{Par-R}]\ \frac{P_1 \xrightarrow{\alpha} P_2}{P \parallel P_1 \xrightarrow{\alpha} P \parallel P_2}$$

$$[\textsc{If-True}]\ \frac{\phi(Eq) = True}{\text{If}\,Eq\,\text{Then}\,P\,\text{Else}\,Q \xrightarrow{\tau} P} \qquad [\textsc{If-False}]\ \frac{\phi(Eq) = False}{\text{If}\,Eq\,\text{Then}\,P\,\text{Else}\,Q \xrightarrow{\tau} Q}$$

$$[\textsc{Seq}]\ \frac{P_1 \xrightarrow{\alpha} P_2}{P_1\,;\,Q \xrightarrow{\alpha} P_2\,;\,Q} \qquad [\textsc{SKIP}]\ \frac{SKIP(P)}{P\,;\,Q \xrightarrow{\tau} Q}$$

Figure 5.2: The LTS of Process
The meta-variables are $c \in \mathcal{C}, z \in \mathcal{V} \cup \mathbb{N}, \alpha \in \mathcal{A}, C \in \mathcal{E}$.

The rule [Seq] permits computing at the left of the symbol ; of the Sequencing.

With the Structural Congruence and the LTS, we can check nice properties like in the previous chapter.

Lemma 5.1.7 (SubstitionRespectsCongruence): $\forall (P, Q) \in \mathcal{P}_{\text{SVCCS}}$, $\forall x \in \mathcal{V}$ and $\forall v \in \mathcal{V} \cup \mathbb{N}$,

$$\text{if } P \equiv Q \text{ then } P[x := v] \equiv Q[x := v]$$

Proof. For the constructors from VCCS, we can see the proof of lemma 4.2.7.

Otherwise :

- for [Sc-sassoc], we have :

  $$P\,;\,(Q\,;\,R) \equiv (P\,;\,Q)\,;\,R$$

  By definition :

  $$(P\,;\,(Q\,;\,R))[x := v] = P[x := v]\,;\,(Q[x := v]\,;\,R[x := v])$$

  Then by [Sc-sassoc] :

  $$P[x := v]\,;\,(Q[x := v]\,;\,R[x := v]) \equiv (P[x := v]\,;\,Q[x := v])\,;\,R[x := v]$$

  And by definition :

  $$(P[x := v]\,;\,Q[x := v])\,;\,R[x := v] = ((P\,;\,Q)\,;\,R)[x := v]$$

- for [Sc-piassoc], we have : :

  $$(\pi\bullet P)\,;\,Q \equiv \pi\bullet(P\,;\,Q) \quad (\pi \in \{\tau\,,\,c\,!\,z\})$$

  By definition :

  $$((\pi\bullet P)\,;\,Q)[x := v] = ((\pi[x := v])\bullet P[x := v])\,;\,Q[x := v]$$

Then by [Sc-piassoc] :

$$((\pi[x := v]) \bullet P[x := v]) \,;\, Q[x := v] \equiv \pi[x := v] \bullet (P[x := v] \,;\, Q[x := v])$$

And by definition :

$$\pi[x := v] \bullet (P[x := v] \,;\, Q[x := v]) = (\pi \bullet (P \,;\, Q))[x := v]$$

- for [Sc-input-assoc], we have :

$$(c\,?\,y \bullet P) \,;\, Q \equiv c\,?\,y \bullet (P \,;\, Q) \quad (y \notin \texttt{FV}(Q))$$

We take the representative with $y \neq x$ and $y \neq v$ to avoid some misunderstandings.

By definition :

$$((c\,?\,y \bullet P) \,;\, Q)[x := v] = (c\,?\,y \bullet P[x := v]) \,;\, Q[x := v]$$

Then by [Sc-input-assoc], we have :

$$(c\,?\,y \bullet P[x := v]) \,;\, Q[x := v] \equiv c\,?\,y \bullet P[x := v] \,;\, Q[x := v]$$

And by definition :

$$c\,?\,y \bullet P[x := v] \,;\, Q[x := v] \equiv (c\,?\,y \bullet P \,;\, Q)[x := v]$$

- for [Sc-ssdist], we have :

$$(G_1 + G_2) \,;\, Q \equiv (G_1 \,;\, Q) + (G_2 \,;\, Q)$$

By definition :

$$((G_1 + G_2) \,;\, Q)[x := v] = (G_1[x := v] + G_2[x := v]) \,;\, Q[x := v]$$

Then by [Sc-ssdist] :

$$(G_1[x := v] + G_2[x := v]) \,;\, Q[x := v] \equiv (G_1[x := v] \,;\, Q[x := v]) + (G_2[x := v] \,;\, Q[x := v])$$

And by definition :

$$(G_1[x := v] \,;\, Q[x := v]) + (G_2[x := v] \,;\, Q[x := v]) = ((G_1 \,;\, Q) + (G_2 \,;\, Q))[x := v]$$

□

Lemma 5.1.8 (CongruenceRespectsTransition): $\forall (S, T) \in \mathcal{P}^2_{\text{SVCCS}},\ \alpha \in \mathcal{A}$, if $S \equiv . \xrightarrow{\alpha} T$ then $S \xrightarrow{\alpha} . \equiv T$.

Proof. We suppose that we have for some S_1 :

$$S \equiv S_1 \xrightarrow{\alpha} T$$

We recall, that we went for an induction on $S \equiv S_1$ in the proof of the lemma 4.2.13.

We can see the proof of the lemma 4.2.13 for the transition rules from VCCS.

Otherwise:

- for [Sc-sassoc], we have by hypothesis :

$$P \,;\, (Q \,;\, R) \equiv (P \,;\, Q) \,;\, R \xrightarrow{\alpha} T$$

Then we have 2 cases :

– if the reduction is made by a SKIP, then we have $SKIP(P)$ and :

$$P\,;\,(Q\,;\,R) \equiv (P\,;\,Q)\,;\,R \xrightarrow{\tau} Q\,;\,R$$

By [SKIP] and reflexivity we have :

$$P\,;\,(Q\,;\,R) \xrightarrow{\tau} Q\,;\,R$$

– in the second case, we have for some P', $P \xrightarrow{\alpha} P'$ and :

$$P\,;\,(Q\,;\,R) \equiv (P\,;\,Q)\,;\,R \xrightarrow{\alpha} (P'\,;\,Q)\,;\,R$$

By [SEQ] and [SC-SASSOC] we have :

$$P\,;\,(Q\,;\,R) \xrightarrow{\alpha} P'\,;\,(Q\,;\,R) \equiv (P'\,;\,Q)\,;\,R$$

■ for the symmetry of [SC-SASSOC], we have by hypothesis for some P', $P \xrightarrow{\alpha} P'$:

$$(P\,;\,Q)\,;\,R \equiv P\,;\,(Q\,;\,R) \xrightarrow{\alpha} P'\,;\,(Q\,;\,R)$$

Then by [SEQ] and [SC-SASSOC] :

$$(P\,;\,Q)\,;\,R \xrightarrow{\alpha} (P'\,;\,Q)\,;\,R \equiv P'\,;\,(Q\,;\,R)$$

■ for [SC-PIASSOC], we have by hypothesis for some $(\pi \in \{\tau\,,\,c\,!\,v\})$, $\pi \bullet P \xrightarrow{\alpha} P$ and :

$$(\pi \bullet P)\,;\,Q \equiv \pi \bullet (P\,;\,Q) \xrightarrow{\alpha} P\,;\,Q$$

Then by [SEQ] and reflexivity, plus [OUTPUT] or [TAU] depending on π:

$$(\pi \bullet P)\,;\,Q \xrightarrow{\alpha} P\,;\,Q \equiv P\,;\,Q$$

■ for the symmetry of [SC-PIASSOC], we have by hypothesis for some $(\pi \in \{\tau\,,\,c\,!\,v\})$, $\pi \bullet P \xrightarrow{\alpha} P$ and :

$$\pi \bullet (P\,;\,Q) \equiv (\pi \bullet P)\,;\,Q \xrightarrow{\alpha} P\,;\,Q$$

Then by [SEQ] and reflexivity, plus [OUTPUT] or [TAU] depending on π:

$$\pi \bullet (P\,;\,Q) \xrightarrow{\alpha} P\,;\,Q \equiv P\,;\,Q$$

■ for [SC-INPUT-ASSOC], we have by hypothesis, $(x \notin \mathtt{FV}(Q))$ and :

$$(c\,?\,x \bullet P)\,;\,Q \equiv c\,?\,x \bullet (P\,;\,Q) \xrightarrow{c\,?\,v} (P\,;\,Q)[x := v]$$

By definition, and by $(x \notin \mathtt{FV}(Q))$, we have :

$$\begin{aligned}(P\,;\,Q)[x := v] &= P[x := v]\,;\,Q[x := v] \\ &= P[x := v]\,;\,Q\end{aligned}$$

Then we have by [INPUT] and reflexivity :

$$(c\,?\,x \bullet P)\,;\,Q \xrightarrow{c\,?\,v} P[x := v]\,;\,Q \equiv P[x := v]\,;\,Q$$

■ for the symmetry of [Sc-input-assoc], we have by hypothesis, $(x \notin \texttt{FV}(Q))$ and :

$$c\,?\,x\,\bullet(P\,;\,Q) \equiv (c\,?\,x\,\bullet P)\,;\,Q \xrightarrow{c\,?\,v} P[x := v]\,;\,Q$$

By definition, and by $(x \notin \texttt{FV}(Q))$, we have :

$$\begin{aligned} P[x := v]\,;\,Q &= P[x := v]\,;\,Q[x := v] \\ &= (P\,;\,Q)[x := v] \end{aligned}$$

Then we have by [Input] and reflexivity :

$$c\,?\,x\,\bullet(P\,;\,Q) \xrightarrow{c\,?\,v} (c\,?\,x\,\bullet P)\,;\,Q \equiv P[x := v]\,;\,Q$$

■ for [Sc-ssdist], we have by hypothesis :

$$(G_1 + G_2)\,;\,Q \equiv (G_1\,;\,Q) + (G_2\,;\,Q) \xrightarrow{\alpha} S$$

Then we have 2 cases :

– if the transition is made by G_1, we have $G_1 \xrightarrow{\alpha} P_1$ and :

$$(G_1 + G_2)\,;\,Q \equiv (G_1\,;\,Q) + (G_2\,;\,Q) \xrightarrow{\alpha} P_1\,;\,Q$$

Then by [Sum-L] and [Seq], we have :

$$(G_1 + G_2)\,;\,Q \equiv P_1\,;\,Q \xrightarrow{\alpha} P_1\,;\,Q$$

– if the transition is made by G_2, we have $G_2 \xrightarrow{\alpha} P_2$ and :

$$(G_1 + G_2)\,;\,Q \equiv (G_1\,;\,Q) + (G_2\,;\,Q) \xrightarrow{\alpha} P_2\,;\,Q$$

Then by [Sum-R] and [Seq], we have :

$$(G_1 + G_2)\,;\,Q \equiv P_2\,;\,Q \xrightarrow{\alpha} P_2\,;\,Q$$

■ for symmetry of [Sc-ssdist], we have by hypothesis :

$$(G_1\,;\,Q) + (G_2\,;\,Q) \equiv (G_1 + G_2)\,;\,Q \xrightarrow{\alpha} S$$

Then we have 2 cases :

– if the transition is made by G_1, we have $G_1 \xrightarrow{\alpha} P_1$ and :

$$(G_1\,;\,Q) + (G_2\,;\,Q) \equiv (G_1 + G_2)\,;\,Q \xrightarrow{\alpha} P_1\,;\,Q$$

Then by [Seq] and [Sum-L], we have :

$$(G_1\,;\,Q) + (G_2\,;\,Q) \equiv P_1\,;\,Q \xrightarrow{\alpha} P_1\,;\,Q$$

– if the transition is made by G_2, we have $G_2 \xrightarrow{\alpha} P_2$ and :

$$(G_1\,;\,Q) + (G_2\,;\,Q) \equiv (G_1 + G_2)\,;\,Q \xrightarrow{\alpha} P_2\,;\,Q$$

Then by [Seq] and [Sum-L], we have :

$$(G_1\,;\,Q) + (G_2\,;\,Q) \equiv P_2\,;\,Q \xrightarrow{\alpha} P_2\,;\,Q$$

■ for [Sc-context], for $C = (\!|)\,;\, R$, we have by hypothesis $P \equiv Q$ and :

$$P\,;\, R \equiv Q\,;\, R \xrightarrow{\alpha} S$$

Then we have 2 cases :

– if the reduction is a $SKIP$ by Q, we have :

$$P\,;\, R \equiv Q\,;\, R \xrightarrow{\alpha} R$$

By construction of $SKIP$ and $P \equiv Q$, we have that P is skipping, then :

$$P\,;\, R \xrightarrow{\alpha} R \equiv R$$

– if the reduction is made by Q, we have $Q \xrightarrow{\alpha} Q'$ and :

$$P\,;\, R \equiv Q\,;\, R \xrightarrow{\alpha} Q'\,;\, R$$

Then we have :

$$P \equiv Q \xrightarrow{\alpha} Q'$$

By induction hypothesis, there exists M such that :

$$P \xrightarrow{\alpha} M \equiv Q'$$

And by [Seq] and [Sc-context], we have :

$$P\,;\, R \xrightarrow{\alpha} M\,;\, R \equiv Q'\,;\, R$$

■ for [Sc-context], for $C = R\,;\, (\!|)$, we have by hypothesis $P \equiv Q$ and :

$$R\,;\, P \equiv R\,;\, Q \xrightarrow{\alpha} S$$

Then we have 2 cases :

– if the reduction is a $SKIP$ by R, we have :

$$R\,;\, P \equiv R\,;\, Q \xrightarrow{\alpha} Q$$

Then by [SKIP] :

$$R\,;\, P \xrightarrow{\alpha} P \equiv Q$$

– if R isn't skipping, then we have R' such that $R \xrightarrow{\alpha} R'$ and by [Seq] and [Sc-context] :

$$R\,;\, P \xrightarrow{\alpha} R'\,;\, P \equiv R'\,;\, Q$$

□

Now we can introduce our MailBox, that will store our values with the addressee, with his State.

Definition 5.1.9 (Mailbox): A mailbox ($\in \mathcal{M}$) is a multiset of $c\,!\,v$ for $c \in \mathcal{C}$ and $v \in Data$.
We note $\{\!\{c\,!\,v\}\!\}$ as the singleton-multiset that contains $c\,!\,v$.

Definition 5.1.10 (States): A State S ($\in \mathcal{S}$) is a pair of a Mailbox and a Process. We note (M, P).

Definition 5.1.11 (Structural Congruence for States): The Structural Congruence over $\mathcal{S}$ is the binary relation defined by :

$$[\textsc{Input}]\ \frac{P \xrightarrow{c\,?\,v} Q}{(M,P) \xrightarrow{c\,?\,v} (M,Q)} \qquad [\textsc{Tau}]\ \frac{P \xrightarrow{\tau} Q}{(M,P) \xrightarrow{\tau} (M,Q)}$$

$$[\textsc{Send}]\ \frac{P \xrightarrow{c\,!\,v} Q}{(M,P) \xrightarrow{\tau} (M \uplus \{\!\{c\,!\,v\}\!\}, Q)} \qquad [\textsc{Output}]\ \frac{}{(M \uplus \{\!\{c\,!\,v\}\!\}, P) \xrightarrow{c\,!\,v} (M,P)}$$

$$[\textsc{Com}]\ \frac{(M,P) \xrightarrow{c\,?\,v} (M',P') \quad (N,Q) \xrightarrow{c\,!\,v} (N',Q')}{(N,P) \xrightarrow{\tau} (N',P')}$$

Figure 5.3: The LTS of States
The meta-variables are $c \in \mathcal{C}, v \in \mathcal{V} \cup \mathbb{N}$.

$$\text{if } P \equiv Q \text{ then for any M, } (M,P) \equiv (M,Q)$$

Definition 5.1.12 (LTS for States): The LTS $(\mathcal{S}, \mathcal{A}, \xrightarrow{\alpha})$ is defined in figure 5.3.

We inherit all the transition from $(\mathcal{P}_{\text{SVCCS}}, \mathcal{A}, \xrightarrow{\alpha})$, at one exception, the output of the LTS of the Processes are now τ-actions.

It's the sending of the message to mailbox.

The rule [Com] is the communication between the MailBox and any Process that can receive via the same channel.

The $c\,!\,v$-transition is now quite clear, in fact, it only checks that the output $c\,!\,v$ is in the MailBox.

Lemma 5.1.13 (CongruenceRespectsTransition): $\forall (S', S'') \in \mathcal{S}^2,\ \alpha \in \mathcal{A}$, if then $S' \xrightarrow{\alpha} . \equiv S''$.

Proof. Take $(S', S'') \in \mathcal{S}^2,\ \alpha \in \mathcal{A}$. Suppose that it exists S_1 such that :

$$S' \equiv S_1 \xrightarrow{\alpha} S''$$

We proceed by induction on $S_1 \xrightarrow{\alpha} S''$.

- for [Input], we have :

  $$(M',P') \equiv (M',P_1) \xrightarrow{c\,?\,v} (M',P'')$$

  Which means, by definition of $\equiv$ and $\xrightarrow{c\,?\,v}$ on States :

  $$P' \equiv P_1 \xrightarrow{c\,?\,v} P''$$

  By lemma 5.1.8, it exists P_1' such that :

  $$P' \xrightarrow{c\,?\,v} P_1' \equiv P''$$

  And then, by [Input] :

  $$(M',P') \xrightarrow{c\,?\,v} (M',P_1') \equiv (M',P'')$$

- for [Output], we have :

  $$(M' \uplus \{\!\{c\,!\,v\}\!\}, P') \equiv (M' \uplus \{\!\{c\,!\,v\}\!\}, P_1) \xrightarrow{c\,!\,v} (M',P_1)$$

  Then by [Output] and reflexivity :

  $$(M' \uplus \{\!\{c\,!\,v\}\!\}, P') \xrightarrow{c\,!\,v} (M',P') \equiv (M',P')$$

- for [TAU], we have :

$$(M', P') \equiv (M', P_1) \xrightarrow{\tau} (M', P'')$$

Which means, by definition of $\equiv$ and $\xrightarrow{\tau}$ by [TAU] on States :

$$P' \equiv P_1 \xrightarrow{\tau} P''$$

By lemma 5.1.8, it exists P_1' such that :

$$P' \xrightarrow{\tau} P_1' \equiv P''$$

And then, by [TAU] :

$$(M', P') \xrightarrow{\tau} (M', P_1') \equiv (M', P'')$$

- for [SEND], we have :

$$(M', P') \equiv (M', P_1) \xrightarrow{\tau} (M' \uplus \{\!\{c\,!\,v\}\!\}, P'')$$

Which means, by definition of $\equiv$ and $\xrightarrow{\tau}$ by [SEND] on States :

$$P' \equiv P_1 \xrightarrow{c\,!\,v} P''$$

By lemma 5.1.8, it exists P_1' such that :

$$P' \xrightarrow{c\,!\,v} P_1' \equiv P''$$

And then, by [TAU] :

$$(M', P') \xrightarrow{\tau} (M' \uplus \{\!\{c\,!\,v\}\!\}, P_1') \equiv (M' \uplus \{\!\{c\,!\,v\}\!\}, P'')$$

- for [COM], we have :

$$(M' \uplus \{\!\{c\,!\,v\}\!\}, P') \equiv (M' \uplus \{\!\{c\,!\,v\}\!\}, P_1) \xrightarrow{\tau} (M', P'')$$

by :

$$P_1 \xrightarrow{c\,?\,v} P''$$

Then by definition of $\equiv$, we have :

$$P' \equiv P_1 \xrightarrow{c\,?\,v} P''$$

By lemma 5.1.8, it exists P_1' such that :

$$P' \xrightarrow{c\,?\,v} P_1' \equiv P''$$

And by [OUTPUT], we have :

$$(M' \uplus \{\!\{c\,!\,v\}\!\}, P') \xrightarrow{c\,!\,v} (M', P')$$

Finally, by [COM] :

$$(M' \uplus \{\!\{c\,!\,v\}\!\}, P') \xrightarrow{\tau} (M', P_1') \equiv (M', P'')$$

□

In our LTS of States, we don't have a concrete rule of [PAR-L] , [PAR-R] or [SEQ], but we inherit all the transition sources and targets of the LTS over $\mathcal{P}_{\text{SVCCS}}$ by the rules [SEND], [TAU] and [INPUT], we can use the same rule for our LTS over States.

LEMMA 5.1.14: $\forall\, (M_1, P_1) \in \mathcal{S}$ and $P \in \mathcal{P}_{\text{SVCCS}}$,

$$\text{if } (M_1, P_1) \xrightarrow{\alpha} (M_2, P_2) \text{ then } (M_1, P_1 \parallel P) \xrightarrow{\alpha} (M_2, P_2 \parallel P)$$

Proof. Take $(M_1, P_1) \in \mathcal{S}$ and $P \in \mathcal{P}_{\text{SVCCS}}$,

Suppose that :

$$(M_1, P_1) \xrightarrow{\alpha} (M_2, P_2)$$

We go for an induction on $(M_1, P_1) \xrightarrow{\alpha} (M_2, P_2)$.

- for [Input], we have :

  $$(M_1, P_1) \xrightarrow{c\,?\,v} (M_1, P_2)$$

  by

  $$P_1 \xrightarrow{c\,?\,v} P_2$$

  Then by [Par-L] :

  $$P_1 \parallel P \xrightarrow{c\,?\,v} P_2 \parallel P$$

  Finally, by [Input] :

  $$(M_1, P_1 \parallel P) \xrightarrow{c\,?\,v} (M_1, P_2 \parallel P)$$

- for [Tau], we have :

  $$(M_1, P_1) \xrightarrow{\tau} (M_1, P_2)$$

  by

  $$P_1 \xrightarrow{\tau} P_2$$

  Then by [Par-L] :

  $$P_1 \parallel P \xrightarrow{c\,?\,v} P_2 \parallel P$$

  Finally, by [Tau] :

  $$(M_1, P_1 \parallel P) \xrightarrow{c\,?\,v} (M_1, P_2 \parallel P)$$

- for [Tau], we have :

  $$(M_1, P_1) \xrightarrow{\tau} (M_1, P_2)$$

  by

  $$P_1 \xrightarrow{\tau} P_2$$

  Then by [Par-L] :

  $$P_1 \parallel P \xrightarrow{c\,?\,v} P_2 \parallel P$$

  Finally, by [Tau] :

  $$(M_1, P_1 \parallel P) \xrightarrow{c\,?\,v} (M_1, P_2 \parallel P)$$

- for [Send], we have :

  $$(M_1, P_1) \xrightarrow{\tau} (M_1 \uplus \{\!\{c\,!\,v\}\!\}, P_2)$$

  by

  $$P_1 \xrightarrow{c\,!\,v} P_2$$

  Then by [Par-L] :

  $$P_1 \parallel P \xrightarrow{c\,!\,v} P_2 \parallel P$$

  Finally, by [Send] :

  $$(M_1, P_1 \parallel P) \xrightarrow{c\,?\,v} (M_1 \uplus \{\!\{c\,!\,v\}\!\}, P_2 \parallel P)$$

- for [Com], we have :

$$(M_2 \uplus \{\!\{c\,!\,v\}\!\}, P_1) \xrightarrow{\tau} (M_2, P_2)$$

by

$$P_1 \xrightarrow{c\,?\,v} P_2$$

Then by [Par-L] :

$$P_1 \parallel P \xrightarrow{c\,?\,v} P_2 \parallel P$$

Finally, by [Com] :

$$(M_2 \uplus \{\!\{c\,!\,v\}\!\}, P_1 \parallel P) \xrightarrow{c\,?\,v} (M_2, P_2 \parallel P)$$

□

Lemma 5.1.15: $\forall\, (M_1, P_1) \in \mathcal{S}$ and $P \in \mathcal{P}_{\text{SVCCS}}$,

$$\text{if } (M_1, P_1) \xrightarrow{\alpha} (M_2, P_2) \text{ then } (M_1, P_1\,;\, P) \xrightarrow{\alpha} (M_2, P_2\,;\, P)$$

Proof. We can see the proof of lemma 5.1.14, but for the rule [Seq] instead of [Par-L]. □

Lemma 5.1.16: $\forall\, (M_1, G_1) \in \mathcal{S}$ and $G_2 \in \mathcal{P}_{\text{SVCCS}}$ such G_1, G_2 are guards,

$$\text{if } (M_1, G_1) \xrightarrow{\tau} (M_2, P_2) \text{ then } (M_1, G_1 + G_2) \xrightarrow{\tau} (M_2, P_2)$$

Proof. Take $(M_1, G_1) \in \mathcal{S}$ and $G_2 \in \mathcal{P}_{\text{SVCCS}}$ such G_1, G_2 are guards.

Suppose that :

$$(M_1, G_1) \xrightarrow{\tau} (M_2, P_2)$$

We go for an induction on $(M_1, G_1) \xrightarrow{\tau} (M_2, P_2)$.

- for [Tau], we have :

$$(M_1, G_1) \xrightarrow{\tau} (M_1, P_2)$$

by $G_1 \xrightarrow{\tau} P_2$.

Then by [Sum-L], we have :

$$G_1 + G_2 \xrightarrow{\tau} P_2$$

Finally, by [Tau] :

$$(M_1, G_1 + G_2) \xrightarrow{\tau} (M_1, P_2)$$

- for [Send], we have :

$$(M_1, G_1) \xrightarrow{\tau} (M_1 \uplus \{\!\{c\,!\,v\}\!\}, P_2)$$

by $G_1 \xrightarrow{c\,!\,v} P_2$.

Then by [Sum-L], we have :

$$G_1 + G_2 \xrightarrow{\tau} P_2$$

Finally, by [Tau] :

$$(M_1, G_1 + G_2) \xrightarrow{\tau} (M_1 \uplus \{\!\{c\,!\,v\}\!\}, P_2)$$

■ for [Com], we have :

$$(M_2 \uplus \{\!\{c\,!\,v\}\!\}, G_1) \xrightarrow{\tau} (M_2, P_2)$$

by $G_1 \xrightarrow{c\,?\,v} P_2$.

Then by [Sum-L], we have :

$$G_1 + G_2 \xrightarrow{c\,?\,v} P_2$$

Finally, by [Com] :

$$(M_2 \uplus \{\!\{c\,!\,v\}\!\}, G_1 + G_2) \xrightarrow{c\,?\,v} (M_2, P_2)$$

□

Those three lemmas will be usefull for the Harmony Lemma.

Now we can look at the STS and another usefull lemma.

Definition 5.1.17 (STS for States): The STS $(\mathcal{S}, \longrightarrow)$ is defined in figure 5.4.

[Sts-com] $\dfrac{}{(M \uplus \{\!\{c\,!\,v\}\!\}, c\,?\,x \bullet P + G) \longrightarrow (M, P[x := v])}$

[Sts-send] $\dfrac{}{(M, c\,!\,v \bullet P + G) \longrightarrow (M \uplus \{\!\{c\,!\,v\}\!\}, P)}$

[Sts-tau] $\dfrac{}{(M, \tau \bullet P + G) \longrightarrow (M, P)}$

[Sts-unf] $\dfrac{}{(M, \mathrm{rec}\, X.C(\!|X|\!)) \longrightarrow (M, C(\!|\mathrm{rec}\, X.C(\!|X|\!)|\!))}$

[Sts-if-True] $\dfrac{\phi(Eq) = True}{(M, \text{If } Eq \text{ Then } P \text{ Else } Q) \longrightarrow (M, P)}$

[Sts-if-False] $\dfrac{\phi(Eq) = False}{(M, \text{If } Eq \text{ Then } P \text{ Else } Q) \longrightarrow (M, Q)}$

[Sts-par] $\dfrac{(M, P) \longrightarrow (M', P')}{(M, P \parallel Q) \longrightarrow (M', P' \parallel Q)}$

[Sts-seq] $\dfrac{(M, P) \longrightarrow (M', P')}{(M, (P\,;\,Q) + G) \longrightarrow (M', P'\,;\,Q)}$

[Sts-SKIP] $\dfrac{SKIP(P)}{(M, P\,;\,Q + G) \longrightarrow (M, Q)}$

[Sts-cong] $\dfrac{S_1 \equiv S_1' \quad S_1' \longrightarrow S_2' \quad S_2' \equiv S_2}{S_1 \longrightarrow S_2}$

Figure 5.4: The STS of States
The meta-variables are $c \in \mathcal{C}, v \in \mathcal{V} \cup \mathbb{N}, \alpha \in \mathcal{A}, C \in \mathcal{E}$.

And we have like the [Sum-L] rule but for the STS :

Lemma 5.1.18: $\forall\, (M_1, G_1) \in S$ and $G_2 \in \mathcal{P}_{\text{SVCCS}}$ such that G_1 and G_2 are guards,

$$\text{if } (M_1, G_1) \longrightarrow (M_2, P_2) \text{ then } (M_1, G_1 + G_2) \longrightarrow (M_2, P_2)$$

Proof. Take $(M_1, G_1) \in S$ and $G_2 \in \mathcal{P}_{\text{SVCCS}}$ such that G_1 and G_2 are guards.

Suppose that :

$$(M_1, G_1) \longrightarrow (M_2, P_2)$$

We go for an induction on $(M_1, G_1) \longrightarrow (M_2, P_2)$.

■ for [Sts-com], we have :

$$(M_2 \uplus \{\!\{c\,!\,v\}\!\}, c\,?\,x \bullet P + G) \longrightarrow (M_2, P[x := v])$$

By [STS-COM] we have :

$$(M_2 \uplus \{\!\{c\,!\,v\}\!\}, c\,?\,x \bullet P + (G + G_2)) \longrightarrow (M_2, P[x := v])$$

Then by [STS-CONG] :

$$(M_2 \uplus \{\!\{c\,!\,v\}\!\}, (c\,?\,x \bullet P + G) + G_2) \longrightarrow (M_2, P[x := v])$$

- for [STS-SEND], we have :

$$(M_1, c\,!\,v \bullet P_2 + G) \longrightarrow (M_1 \uplus \{\!\{c\,!\,v\}\!\}, P_2)$$

By [STS-SEND], we have :

$$(M_1, c\,!\,v \bullet P_2 + (G + G_2)) \longrightarrow (M_1 \uplus \{\!\{c\,!\,v\}\!\}, P_2)$$

Then by [STS-CONG] :

$$(M_1, (c\,!\,v \bullet P_2 + G) + G_2) \longrightarrow (M_1 \uplus \{\!\{c\,!\,v\}\!\}, P_2)$$

- for [STS-TAU], we have :

$$(M_1, \tau \bullet P_2 + G) \longrightarrow (M_1, P_2)$$

By [STS-TAU], we have :

$$(M_1, \tau \bullet P_2 + (G + G_2)) \longrightarrow (M_1, P_2)$$

Then by [STS-CONG] :

$$(M_1, (\tau \bullet P_2 + G) + G_2) \longrightarrow (M_1, P_2)$$

- for [STS-SEQ], we have :

$$(M, (P\,;\,Q) + G) \longrightarrow (M', P'\,;\,Q)$$

by $(M, P) \longrightarrow (M', P')$.

By [STS-SEQ], we have :

$$(M, (P\,;\,Q) + (G + G_2)) \longrightarrow (M', P'\,;\,Q)$$

by $(M, P) \longrightarrow (M', P')$.

Then by [STS-CONG] :

$$(M, ((P\,;\,Q) + G) + G_2) \longrightarrow (M', P'\,;\,Q)$$

- for [STS-SKIP], we have :

$$(M_2, (P\,;\,P_2) + G) \longrightarrow (M_2, P_2)$$

by $SKIP(P)$.

By [STS-SEQ], we have :

$$(M_2, (P\,;\,P_2) + (G + G_2)) \longrightarrow (M_2, P_2)$$

Then by [STS-CONG] :

$$(M_2, ((P\,;\,P_2) + G) + G_2) \longrightarrow (M_2, P_2)$$

- for [STS-CONG], we have :

$$(M_1, G_1) \longrightarrow (M_2, P_2)$$

by $(M_1, G_1) \equiv (M_1, G_1') \longrightarrow (M_2, P_2') \equiv (M_2, P_2)$.

Then by induction hypothesis, we have :

$$(M_1, G_1' + G_2) \longrightarrow (M_2, P_2')$$

By [Sc-context], we have :

$$G_1 + G_2 \equiv G_1' + G_2$$

Finally, by [Sts-cong], we have :

$$(M_1, G_1 + G_2) \longrightarrow (M_2, P_2)$$

□

5.2 Harmony Lemma and Asynchrony

We don't have the lemma 4.2.8. Indeed we can alternate the sequencing and the parallel for a reduction.

So, we can't use it for the proof of the Harmony Lemma.

Lemma 5.2.1 (Harmony Lemma): $\forall (S, T) \in \mathcal{S}^2$,

$$S \longrightarrow T \text{ if and only if } S \xrightarrow{\tau} . \equiv T$$

Proof. Take $(S, T) \in \mathcal{S}^2$, We have two side to prove :

- if $S \longrightarrow T$ then $S \xrightarrow{\tau} . \equiv T$
- if $S \xrightarrow{\tau} . \equiv T$ then $S \longrightarrow T$

We go for an induction on the reduction/transition depending on each side.

- For the first side, we go for an induction on $S \longrightarrow T$.
  - for [Sts-com], we have :

    $$(M \uplus \{\!\{c\,!\,v\}\!\}, c\,?\,x \bullet P + G) \longrightarrow (M, P[x := v])$$

    By [Input], we have :

    $$c\,?\,x \bullet P \xrightarrow{c\,?\,v} P[x := v]$$

    And by [Sum-L] :

    $$c\,?\,x \bullet P + G \xrightarrow{c\,?\,v} P[x := v]$$

    Then by [Com] and reflexivity, we have :

    $$(M \uplus \{\!\{c\,!\,v\}\!\}, c\,?\,x \bullet P + G) \xrightarrow{\tau} (M, P[x := v]) \equiv \xrightarrow{\tau} (M, P[x := v])$$

  - for [Sts-send], we have :

    $$(M, c\,!\,v \bullet P + G) \longrightarrow (M \uplus \{\!\{c\,!\,v\}\!\}, P)$$

    By [Output], we have :

    $$c\,!\,v \bullet P \xrightarrow{c\,!\,v} P$$

    And by [Sum-L] :

    $$c\,!\,x \bullet P + G \xrightarrow{c\,!\,v} P$$

    Then by [Send] and reflexivity, we have :

    $$(M, c\,!\,x \bullet P + G) \xrightarrow{\tau} (M \uplus \{\!\{c\,!\,v\}\!\}, P) \equiv (M \uplus \{\!\{c\,!\,v\}\!\}, P)$$

– for [Sts-send], we have :

$$(M, c\,!\,v \bullet P + G) \longrightarrow (M \uplus \{\!\{c\,!\,v\}\!\}, P)$$

By [Output], we have :

$$c\,!\,v \bullet P \xrightarrow{c\,!\,v} P$$

And by [Sum-L] :

$$c\,!\,x \bullet P + G \xrightarrow{c\,!\,v} P$$

Then by [Send] and reflexivity, we have :

$$(M, c\,!\,x \bullet P + G) \xrightarrow{\tau} (M \uplus \{\!\{c\,!\,v\}\!\}, P) \equiv (M \uplus \{\!\{c\,!\,v\}\!\}, P)$$

– for [Sts-tau], we have :

$$(M, \tau \bullet P + G) \longrightarrow (M, P)$$

By [Tau], we have :

$$\tau \bullet P \xrightarrow{\tau} P$$

And by [Sum-L] :

$$\tau \bullet P + G \xrightarrow{\tau} P$$

Then by [Tau] and reflexivity, we have :

$$(M, \tau \bullet P + G) \xrightarrow{\tau} (M, P) \equiv (M, P)$$

– for [Sts-unf], we have :

$$(M, \text{rec}\, X.C(\!|X|\!)) \longrightarrow (M, C(\!|\text{rec}\, X.C(\!|X|\!)|\!))$$

By [Unf], we have :

$$\text{rec}\, X.C(\!|X|\!) \xrightarrow{\tau} C(\!|\text{rec}\, X.C(\!|X|\!)|\!)$$

Then by [Tau] and reflexivity, we have :

$$(M, \text{rec}\, X.C(\!|X|\!)) \xrightarrow{\tau} (M, C(\!|\text{rec}\, X.C(\!|X|\!)|\!)) \equiv (M, C(\!|\text{rec}\, X.C(\!|X|\!)|\!))$$

– for [Sts-if-True], we have :

$$(M, \text{If}\, Eq\, \text{Then}\, P\, \text{Else}\, Q) \longrightarrow (M, P)$$

by $\phi(Eq) = True$.

By [If-True], we have :

$$\text{If}\, Eq\, \text{Then}\, P\, \text{Else}\, Q \xrightarrow{\tau} P$$

Then by [Tau] and reflexivity, we have :

$$(M, \text{If}\, Eq\, \text{Then}\, P\, \text{Else}\, Q) \xrightarrow{\tau} (M, P) \equiv (M, P)$$

– for [Sts-if-False], we have :

$$(M, \text{If}\, Eq\, \text{Then}\, P\, \text{Else}\, Q) \longrightarrow (M, Q)$$

by $\phi(Eq) = False$.

By [If-False], we have :

$$\text{If}\, Eq\, \text{Then}\, P\, \text{Else}\, Q \xrightarrow{\tau} Q$$

Then by [Tau] and reflexivity, we have :

$$(M, \text{If}\, Eq\, \text{Then}\, P\, \text{Else}\, Q) \xrightarrow{\tau} (M, Q) \equiv (M, Q)$$

– for [Sts-par], we have :

$$(M, P \parallel Q) \longrightarrow (M', P' \parallel Q)$$

by $(M, P) \longrightarrow (M', P')$.

By induction hypothesis, it exists P'' such that :

$$(M, P) \xrightarrow{\tau} (M', P'') \equiv (M', P')$$

By the lemma 5.1.14, we have :

$$(M, P \parallel Q) \xrightarrow{\tau} (M', P'' \parallel Q) \equiv (M', P'' \parallel Q)$$

– for [Sts-SKIP], we have :

$$(M, P\,;\, Q + G) \longrightarrow (M, Q)$$

by P skipping.

By [SKIP], we have :

$$P\,;\, Q \xrightarrow{\tau} Q$$

By [Sum-L], we have :

$$P\,;\, Q + G \xrightarrow{\tau} Q$$

Then by [Tau] and reflexivity, we have :

$$(M, P\,;\, Q + G) \xrightarrow{\tau} (M, Q)$$

– for [Sts-seq], we have :

$$(M, (P\,;\, Q) + G) \longrightarrow (M', P'\,;\, Q)$$

by $(M, P) \longrightarrow (M', P')$.

By induction hypothesis, it exists P'' such that :

$$(M, P) \xrightarrow{\tau} (M', P'') \equiv (M', P')$$

Then by lemma 5.1.15, we have :

$$(M, P\,;\, Q) \xrightarrow{\tau} (M', P''\,;\, Q)$$

Finally by lemma 5.1.16, we have :

$$(M, (P\,;\, Q) + G) \xrightarrow{\tau} (M', P''\,;\, Q)$$

– for [Sts-cong], we have :

$$S_1 \longrightarrow S_2$$

by $S_1 \equiv S_1' \longrightarrow S_2' \equiv S_2$

By induction hypothesis, it exists S'' such that :

$$S_1' \xrightarrow{\tau} S'' \equiv S_2'$$

Then by the lemma 5.1.13, it exists S_3 such that :

$$S_1 \xrightarrow{\tau} S_3 (\equiv S'' \equiv S_2') \equiv S_2$$

■ For the second side, we go for an induction on $S \xrightarrow{\tau} T$.

Then we have 3 cases :

☒ for [Tau], we have :

$$(M, P) \xrightarrow{\tau} (M, Q)$$

by $P \xrightarrow{\tau} Q$.

We have then to reason with an induction on $P \xrightarrow{\tau} Q$:

* for $\tau \bullet Q \xrightarrow{\tau} Q$, we have by [Sts-tau] :

$$(M, \tau \bullet Q + \delta) \longrightarrow (M, Q)$$

And by [Sts-cong] :

$$(M, \tau \bullet Q) \longrightarrow (M, Q)$$

* for [Unf], we have by [Sts-unf] :

$$(M, \text{rec } X.C(\!|X|\!)) \longrightarrow (M, C(\!|\text{rec } X.P|\!))$$

* for [If-True] and $\phi(Eq) = True$, we have by [Sts-if-True]:

$$(M, \text{If } Eq \text{ Then } Q \text{ Else } P_1) \xrightarrow{\tau} (M, Q)$$

* for [If-False] and $\phi(Eq) = False$, we have by [Sts-if-False]:

$$(M, \text{If } Eq \text{ Then } P_1 \text{ Else } Q) \xrightarrow{\tau} (M, Q)$$

* for [Seq], we have by [Sts-seq] :

$$(M, P_1 \,;\, P_2 + \delta) \longrightarrow (M, P_1' \,;\, P_2)$$

And by [Sts-cong] :

$$(M, P_1 \,;\, P_2) \longrightarrow (M, P_1' \,;\, P_2)$$

* for [Par-L], we conclude by induction hypothesis.
* for [Par-R], then by [Sts-cong] with [Sc-pcom] and induction hypothesis, we conclude.
* for [Sum-L], we use the result from lemma 5.1.16 and conclude by induction hypothesis.
* for [Sum-R], we use the result from lemma 5.1.16 and conclude by induction hypothesis and [Sts-cong] with [Sc-scom].
* for [SKIP], we conclude with [Sts-SKIP] and [Sts-cong] via [Sc-sneut].

☒ for [Send] we have :

$$(M, P) \xrightarrow{\tau} (M \uplus \{\!\{c\,!\,v\}\!\}, Q)$$

by $P \xrightarrow{c\,!\,v} Q$.

We have then to reason with an induction on $P \xrightarrow{c\,!\,v} Q$:

* for [Output], we have our result by [Sts-send] and [Sts-cong] with [Sc-sneut].
* for [Par-L], we conclude by [Sts-par] and induction hypothesis.
* for [Par-R], we conclude by [Sts-par] and [Sts-cong] via [Sc-pcom] with induction hypothesis.
* for [Sum-L], we use the result from lemma 5.1.16 and conclude by induction hypothesis.

- for [Sum-R], we use the result from lemma 5.1.16 and conclude by induction hypothesis and [Sts-cong] with [Sc-scom].
- for [Seq], we have our result by [Sts-seq] and [Sts-cong] with [Sc-sneut].

☒ for [Com] we have :

$$(N' \uplus \{\!\{c\,?\,v\}\!\}, P) \xrightarrow{\tau} (N', P')$$

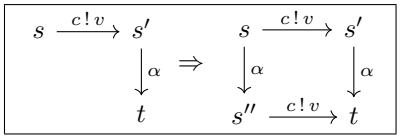
by $P \xrightarrow{c\,?\,v} P'$.

We have then to reason with an induction on $P \xrightarrow{c\,?\,v} P'$:

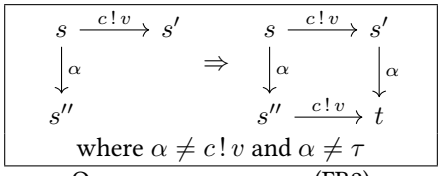
- for [Input], we have our result by [Sts-com] and [Sts-cong] with [Sc-sneut].
- for [Par-L], we conclude by [Sts-par] and induction hypothesis.
- for [Par-R], we conclude by [Sts-par] and [Sts-cong] via [Sc-pcom] with induction hypothesis.
- for [Sum-L], we use the result from lemma 5.1.16 and conclude by induction hypothesis.
- for [Sum-R], we use the result from lemma 5.1.16 and conclude by induction hypothesis and [Sts-cong] with [Sc-scom].
- for [Seq], we have our result by [Sts-seq] and [Sts-cong] with [Sc-sneut].

□

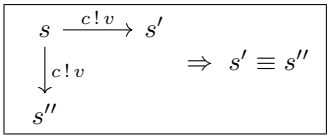
We'll see now the one of the advantages by taking a Mailbox (multiset) to store the output. All of the proof of the axioms for out buffered agents with feedback become more simple.



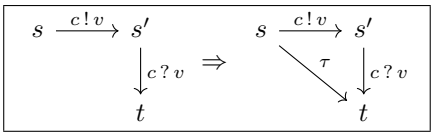
Output-commutativity (FB1)



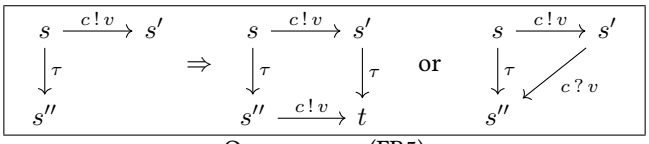
Output-confluence (FB2)



Output-determinacy (FB3)



Feedback (FB4)



Output-tau (FB5)

Output-commutativity (FB1). Assume that we have :

$$s \xrightarrow{c!v} s' \downarrow\alpha\ t$$

Then by definition we have :

$$(M \uplus \{\!\{c!v\}\!\}, P) \xrightarrow{c!v} (M, P) \downarrow\alpha\ (M'', P'')$$

Then :

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{c\,!\,v} & (M, P) \\ \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \alpha} \\ (M'' \uplus \{\!\{c\,!\,v\}\!\}, P'') & \xrightarrow{c\,!\,v} & (M'', P'') \end{array}$$

□

Output-confluence (FB2). Suppose that $\alpha \neq c\,!\,v$, $\alpha \neq \tau$ and

$$\begin{array}{ccc} s & \xrightarrow{c\,!\,v} & s' \\ \downarrow{\scriptstyle \alpha} & & \\ s'' & & \end{array}$$

By definition, we have :

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{c\,!\,v} & (M, P) \\ \downarrow{\scriptstyle \alpha} & & \\ (M'', P'') & & \end{array}$$

- if $\alpha = c'\,!\,v'$, then we have :

$$\begin{array}{ccc} (M''' \uplus \{\!\{c\,!\,v\}\!\} \uplus \{\!\{c'\,!\,v'\}\!\}, P) & \xrightarrow{c\,!\,v} & (M''' \uplus \{\!\{c'\,!\,v'\}\!\}, P) \\ \downarrow{\scriptstyle c'\,!\,v'} & & \\ (M''' \uplus \{\!\{c\,!\,v\}\!\}, P) & & \end{array}$$

  Then :

$$\begin{array}{ccc} (M''' \uplus \{\!\{c\,!\,v\}\!\} \uplus \{\!\{c'\,!\,v'\}\!\}, P) & \xrightarrow{c\,!\,v} & (M''' \uplus \{\!\{c'\,!\,v'\}\!\}, P) \\ \downarrow{\scriptstyle c'\,!\,v'} & & \downarrow{\scriptstyle c'\,!\,v'} \\ (M''' \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{\quad c\,!\,v \quad} & (M''', P) \end{array}$$

- if $\alpha = c\,?\,v$, then we have $P' \xrightarrow{c\,?\,v} P''$ and :

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{c\,!\,v} & (M, P') \\ \downarrow{\scriptstyle c\,?\,v} & & \\ (M \uplus \{\!\{c\,!\,v\}\!\}, P'') & & \end{array}$$

  Then :

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{c\,!\,v} & (M, P') \\ \downarrow{\scriptstyle c\,?\,v} & & \downarrow{\scriptstyle c\,?\,v} \\ (M \uplus \{\!\{c\,!\,v\}\!\}, P'') & \xrightarrow{c\,!\,v} & (M, P'') \end{array}$$

□

Output-determinacy (FB3). Assume that we have :

$$\begin{array}{ccc} s & \xrightarrow{c\,!\,v} & s' \\ \downarrow{\scriptstyle c\,!\,v} & & \\ s'' & & \end{array}$$

By definition :

$$\begin{array}{ccc} & (M \uplus \{\!\{c\,!\,v\}\!\} \uplus \{\!\{c\,!\,v\}\!\}, P) & \\ {\scriptstyle c\,!\,v}\swarrow & & \searrow{\scriptstyle c\,!\,v} \\ (M \uplus \{\!\{c\,!\,v\}\!\}, P) & = & (M \uplus \{\!\{c\,!\,v\}\!\}, P) \end{array}$$

□

Feedback (FB4). Assume that we have :

$$\begin{array}{ccc} s & \xrightarrow{c\,!\,v} & s' \\ & & \downarrow{\scriptstyle c\,?\,v} \\ & & t \end{array}$$

By definition, we have $P \xrightarrow{c\,?\,v} P''$:

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{c\,!\,v} & (M, P) \\ & & \downarrow{\scriptstyle c\,?\,v} \\ & & (M, P'') \end{array}$$

Then by [Com], we have :

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{c\,!\,v} & (M, P) \\ & {\scriptstyle \tau}\searrow & \downarrow{\scriptstyle c\,?\,v} \\ & & (M, P'') \end{array}$$

□

Output-tau (FB5). Assume that we have :

$$\begin{array}{ccc} s & \xrightarrow{c\,!\,v} & s' \\ \downarrow{\scriptstyle \tau} & & \\ s'' & & \end{array}$$

Then by definition, we have :

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{c\,!\,v} & (M, P) \\ \downarrow{\scriptstyle \tau} & & \\ (M'', P'') & & \end{array}$$

Then we have 3 cases for τ :

- if the reduction is made by [Tau], we have $P \xrightarrow{\tau} P''$ and :

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{c\,!\,v} & (M, P) \\ \downarrow \tau & & \\ (M \uplus \{\!\{c\,!\,v\}\!\}, P'') & & \end{array}$$

Then we have :

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{c\,!\,v} & (M, P) \\ \downarrow \tau & & \downarrow \tau \\ (M \uplus \{\!\{c\,!\,v\}\!\}, P'') & \xrightarrow{c\,!\,v} & (M, P'') \end{array}$$

- if the reduction is made by [SEND], we have $P \xrightarrow{c'\,!\,v'} P''$ and :

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{\quad c\,!\,v \quad} & (M, P) \\ \downarrow \tau & & \\ (M \uplus \{\!\{c\,!\,v\}\!\} \uplus \{\!\{c'\,!\,v'\}\!\}, P'') & & \end{array}$$

Then we have :

$$\begin{array}{ccc} (M \uplus \{\!\{c\,!\,v\}\!\}, P) & \xrightarrow{\quad c\,!\,v \quad} & (M, P) \\ \downarrow \tau & & \downarrow \tau \\ (M \uplus \{\!\{c\,!\,v\}\!\} \uplus \{\!\{c'\,!\,v'\}\!\}, P'') & \xrightarrow{c\,!\,v} & (M \uplus \{\!\{c'\,!\,v'\}\!\}, P'') \end{array}$$

□

$r\,?$

$r\,!$

τ

$\tau \bullet r_2\,!\,\textcircled{0}$

$\tau \bullet \tau \bullet r_1\,!\,\textcircled{0}$

$\tau \bullet \tau \bullet \textcircled{1}$

$\xrightarrow{r\,?}$

$\xrightarrow{r\,!}$

rec $X.(Client\,!\,Glace \bullet \textcircled{0} + X)$

rec $X.c\,?\,x \bullet$If $0 < 4$ Then $Printer\,!\,\overline{hello} \bullet \textcircled{0} \parallel X$ Else $\textcircled{0} \parallel c\,!\,0 \bullet c\,!\,1 \bullet c\,!\,2 \bullet c\,!\,3 \bullet \textcircled{0}$

$c\,!\,v \bullet \textcircled{0}$

$\delta\,\textcircled{0}$

$c\,?\,x \bullet InstructionSuivante(x)$

Appendix A

Coq implementation

As said, all the theory has been implemented in COQ.

It is available at `https://github.com/GLmaths/testing-theory.git`.

We will talk here more what was made for the COQ implementation like the De Brujin indices for processes.

We will take the file $VACCS\_SEQ\_Instance.v$ as an example.

A.1 De Brujin indices

To use free and bound variable we use the De Brujin indices.

It bounds a variable to an input abstraction, through his depth.

```
44 Inductive Data :=
45 | cst : Value -> Data
46 | bvar : nat -> Data. (* variable as De Bruijn indices *).
```

```
(* Definition of processes*)
Inductive proc : Type :=
(* To parallele two processes*)
| pr_par : proc -> proc -> proc
(* Variable in a process, for recursion and substitution*)
| pr_var : nat -> proc
(* recursion for process*)
| pr_rec : nat -> proc -> proc
(* If test *NEW term in comparison of CCS* *)
| pr_if_then_else : Equation Data -> proc -> proc -> proc
(* The Process that does nothing without blocking state*)
| pr_nil : proc
(*The Guards*)
| g : gproc -> proc

with gproc : Type :=
(* Deadlock, no more computation *)
| gpr_deadlock : gproc
(* The Success operation*)
| gpr_success : gproc
(*An input is a name of a channel, an input variable, followed by a process*)
| gpr_input : Channel ->  proc -> gproc
(*An output is a name of a channel, an ouput value, followed by a process*)
| gpr_output : Channel -> Data -> proc -> gproc
(*A tau action : does nothing *)
| gpr_tau : proc -> gproc
(* To choose between two processes*)
| gpr_choice : gproc -> gproc -> gproc
(*Sequentiality for process*)
| gpr_seq : proc -> proc -> gproc
.
```

The "cst" constructor is essentially here to take a set of concrete data, to do not use an encoded number like Church numbers.

If we take the term :

$$c\,?\,y \bullet \text{If } x > 50 \text{ Then } \delta \text{ Else } ①$$

Then we can take it for the Input constructor :

$$c\,?\,x \bullet c\,?\,y \bullet \text{If } x > 50 \text{ Then } \delta \text{ Else } ①$$

And the "x" is automatically bound.

But with the Brujin Indices, we have to take care which natural we attribute to the constructor "bvar", to take care of the right depth.

Indeed we can't take the term, for example :

$$\text{If } (bvar\ 0) > 50 \text{ Then } \delta \text{ Else } ①$$

Because it will link the "$bvar\ 0$" to the first input abstraction.

So unlike the term with α-equivalence. We have to take care how we will link all the Input "in advance".

Before defining the substitution, we have to introduce a function that will lift a "bvar" wil it will cross a Input abstraction.

For example, if we have, with α-equivalence :

$$P := c\,?\,y \bullet \text{If } x > 10 \text{ Then } \delta \text{ Else } ①[y := z]$$

Then, we have :

$$P = c\,?\,y \bullet \text{If } z > 10 \text{ Then } \delta \text{ Else } ①$$

But with De Brujin indices, if we do not lift the "z" variable, assume it is $bvar\ 0$ with De Brujin Indices, then we have :

$$P := c\,?\,\bullet \text{If } bvar\ 0 > 10 \text{ Then } \delta \text{ Else } ①$$

And now the variable "$bvar\ 0$" in the If-condition, is bound to $c\,?$.

It should be :

$$c\,?\,\bullet \text{If } bvar\ 1 > 10 \text{ Then } \delta \text{ Else } ①$$

We have to lift it through each Input we crossed.

```
143 Fixpoint NewVar_in_Equation (k : nat) (E : Equation Data) : Equation Data :=
144 match E with
145 | tt => tt
146 | ff => ff
147 | D1 ⩽ D2 => (NewVar_in_Data k D1) ⩽ (NewVar_in_Data k D2)
148 | e1 ∨ e2 => (NewVar_in_Equation k e1) ∨ (NewVar_in_Equation k e2)
149 | non e => non (NewVar_in_Equation k e)
150 end.
151
152 Fixpoint NewVar (k : nat) (p : proc) {struct p} : proc :=
153 match p with
154 | P ‖ Q => (NewVar k P) ‖ (NewVar k Q)
155 | pr_var i => pr_var i
156 | rec x • P =>  rec x • (NewVar k P)
157 | If C Then P Else Q => If (NewVar_in_Equation k C) Then (NewVar k P) Else (NewVar k Q)
158 | 𝟘 => 𝟘
159 | g M => gNewVar k M
160 end
161
162 with gNewVar k M {struct M} : gproc :=
163 match M with
164 | δ => δ
165 | ① => ①
166 | c ? x • p => c ? x • (NewVar (S k) p)
167 | c ! v • p => c ! (NewVar_in_Data k v) • (NewVar k p)
168 | t • p => t • (NewVar k p)
169 | p1 + p2 => (gNewVar k p1) + (gNewVar k p2)
170 | P ; Q => (NewVar k P) ; (NewVar k Q)
171 end.
```

And now we can define the substitution :

```
212 (*Definition of the Substitution *)
213 Definition subst_Data (k : nat) (X : Data) (Y : Data) : Data :=
214 match Y with
215 | cst v => cst v
216 | bvar i => if (decide(i = k)) then X else if (decide(i < k)) then bvar i
217                                                               else bvar (Nat.pred i)
218 end.
219
220 Definition Succ_bvar (X : Data) : Data :=
221 match X with
222 | cst v => cst v
223 | bvar i => bvar (S i)
224 end.
225
226 Fixpoint subst_in_Equation (k : nat) (X : Data) (E : Equation Data) : Equation Data :=
227 match E with
228 | tt => tt
229 | ff => ff
230 | D1 ⩽ D2 => (subst_Data k X D1) ⩽ (subst_Data k X D2)
231 | e1 ∨ e2 => (subst_in_Equation k X e1) ∨ (subst_in_Equation k X e2)
232 | non e => non (subst_in_Equation k X e)
233 end.
234
235 Fixpoint subst_in_proc (k : nat) (X : Data) (p : proc) {struct p} : proc :=
236 match p with
237 | P ‖ Q => (subst_in_proc k X P) ‖ (subst_in_proc k X Q)
238 | pr_var i => pr_var i
239 | rec x • P =>  rec x • (subst_in_proc k X P)
240 | If C Then P Else Q => If (subst_in_Equation k X C) Then (subst_in_proc k X P) Else (subst_in_proc k X Q)
241 | 0 => 0
242 | g M => subst_in_gproc k X M
243 end
244
245 with subst_in_gproc k X M {struct M} : gproc :=
246 match M with
247 | δ => δ
248 | ① => ①
249 | c ? x • p => c ? x • (subst_in_proc (S k) (NewVar_in_Data 0 X) p)
250 | c ! v • p => c ! (subst_Data k X v) • (subst_in_proc k X p)
251 | t • p => t • (subst_in_proc k X p)
252 | p1 + p2 => (subst_in_gproc k X p1) + (subst_in_gproc k X p2)
253 | P ; Q => (subst_in_proc k X P) ; (subst_in_proc k X Q)
254 end.
255
256 Notation "t1 ^ x1" := (subst_in_proc 0 x1 t1).
```

The interesting part is the "$subst\_Data$".

We lift the variable that we substituted to match his link, but when we did the substitution, we destroyed a Input $c\,?$ abstractor.

Then we have to check if the Input $c\,?$ abstractor that we destroyed was needed in any bvar in the sub-term of $c\,?$:

```
if (decide(i < k)) then bvar i
                   else bvar (Nat.pred i)
```

We decide to duplicate the rules of the congruence in the definition to "gain" symmetry by definition and then to simplify the proofs.

```
Inductive cgr_step : proc -> proc -> Prop :=
(*  Reflexivity of the Relation ≡  *)
| cgr_refl_step : forall p, p ≡ p

(* Rules for the Parallèle *)
| cgr_par_com_step : forall p q,
    p ‖ q ≡ q ‖ p
| cgr_par_assoc_step : forall p q r,
    (p ‖ q) ‖ r ≡ p ‖ (q ‖ r)
| cgr_par_assoc_rev_step : forall p q r,
    p ‖ (q  ‖ r) ≡ (p ‖ q) ‖ r

(* Rules for the Summation *)
| cgr_choice_nil_step : forall p,
    cgr_step (p + δ) p
| cgr_choice_nil_rev_step : forall p,
    cgr_step (g p) (p + δ)
| cgr_choice_com_step : forall p q,
    cgr_step (p + q) (q + p)
| cgr_choice_assoc_step : forall p q r,
    cgr_step ((p + q) + r) (p + (q + r))
| cgr_choice_assoc_rev_step : forall p q r,
    cgr_step (p + (q + r)) ((p + q) + r)

(*The reduction is given to certain terms...*)
| cgr_recursion_step : forall x p q,
    cgr_step p q -> (rec x • p) ≡ (rec x • q)
| cgr_tau_step : forall p q,
    cgr_step p q ->
    cgr_step (t • p) (t • q)
| cgr_input_step : forall c p q,
    cgr_step p q ->
    cgr_step (c ? x • p) (c ? x • q)
| cgr_output_step : forall c v p q,
    cgr_step p q ->
    cgr_step (c ! v • p) (c ! v • q)
| cgr_par_step : forall p q r,
    cgr_step p q ->
    p ‖ r ≡ q ‖ r
| cgr_if_left_step : forall C p q q',
    cgr_step q q' ->
    (If C Then p Else q) ≡ (If C Then p Else q')
| cgr_if_right_step : forall C p p' q,
    cgr_step p p' ->
    (If C Then p Else q) ≡ (If C Then p' Else q)

(*...and sums (only for guards (by sanity))*)
| cgr_choice_step : forall p1 q1 p2,
    cgr_step (g p1) (g q1) ->
    cgr_step (p1 + p2) (q1 + p2)
| cgr_seq_left_step : forall p q r,
    cgr_step p q ->
    (g (p ; r)) ≡ (g (q ; r ))
| cgr_seq_right_step : forall p q r,
    cgr_step p q ->
    (g (r ; p)) ≡ (g (r ; q))
....
(* Rules for sequentiation *)
(*| cgr_seq_nil_step : forall P, cgr_step (δ ; P) δ
| cgr_seq_nil_rev_step : forall P, cgr_step δ (δ ; P)*)
| cgr_seq_assoc_step : forall p q r,
    cgr_step ((p ; q) ; r) (p ; (q ; r))
| cgr_seq_assoc_rev_step : forall p q r,
    cgr_step (p ; (q ; r)) ((p ; q) ; r)

(* Rules for sequentiation-analysis *)
| cgr_seq_choice_step : forall G G' P, cgr_step ((G + G') ; P) ((G ; P) + (G'; P))
| cgr_seq_choice_rev_step : forall G G' P, cgr_step (((g G) ; P) + ((g G'); P)) ((G + G') ; P)
| cgr_seq_tau_step : forall P Q, cgr_step ((t • P) ; Q) (t • (P ; Q))
| cgr_seq_tau_rev_step : forall P Q, cgr_step (t • (P ; Q)) ((t • P) ; Q)
| cgr_seq_input_step : forall c P Q , cgr_step ((c ? x • P) ; Q) (c ? x • (P ; NewVar 0 Q))
| cgr_seq_input_rev_step : forall c P Q , cgr_step (c ? x • (P ; NewVar 0 Q)) ((c ? x • P) ; Q)
| cgr_seq_output_step : forall c v P Q, cgr_step ((c ! v • P) ; Q) (c ! v • (P ; Q))
| cgr_seq_output_rev_step : forall c v P Q, cgr_step (c ! v • (P ; Q)) ((c ! v • P) ; Q)
.
```

Then we take his transitive closure :

```
(* Defining the transitive closure of ≡ *)
Definition cgr := (clos_trans proc cgr_step).
```

The most interesting rule is the rule (with α-equivalence) :

$$(c\,?\,x\,\bullet P)\,;\,Q \equiv c\,?\,x\,\bullet(P\,;\,Q) \quad (x \notin \mathtt{FV}(Q))$$

In COQ :

```
490 | cgr_seq_input_step : forall c P Q , cgr_step ((c ? x • P) ; Q) (c ? x • (P ; NewVar 0 Q))
```

The role of "NewVar" is to lift the variable in the process that are bounded with a outside Input from the actual Input in the rule.

This is needed because the congruence is closed by context

For example, take the process :

$$c'\,?\,\bullet(c\,?\,\bullet \text{If } bvar\ 0 > 50 \text{ Then } \delta \text{ Else } \textcircled{1}\,;\,c\,!\,bvar\ 0\ \bullet\textcircled{0})$$

The $bvar\ 0$, in "If $bvar\ 0 > 50$ Then δ Else $\textcircled{1}$", is linked to $c\,?$ and the $bvar\ 0$, in "$c\,!\,bvar\ 0\ \bullet\textcircled{0}$", is linked to $c'\,?$.

If we apply the congruence rule for :

$$c\,?\,\bullet \text{If } bvar\ 0 > 50 \text{ Then } \delta \text{ Else } \textcircled{1}\,;\,c\,!\,bvar\ 0\ \bullet\textcircled{0}$$

We have to adapt the second $bvar\ 0$, it has be linked after the congruence rule with the same Input :

$$c\,?\,\bullet(\text{If } bvar\ 0 > 50 \text{ Then } \delta \text{ Else } \textcircled{1}\,;\,c\,!\,bvar\ 1\ \bullet\textcircled{0})$$

And then with the context $C = c'\,?\,\bullet(\!|\,|\!)$:

$$c'\,?\,\bullet c\,?\,\bullet(\text{If } bvar\ 0 > 50 \text{ Then } \delta \text{ Else } \textcircled{1}\,;\,c\,!\,bvar\ 1\ \bullet\textcircled{0})$$

The location of the $bvar$ is still linked to $c\,?$.

Like we said in the main document, we proved the Harmony Lemma :

```
1626 Theorem HarmonyLemmaForCCSWithValuePassing : forall P Q, (ltsM_then_sc P τ Q) <-> (sts P Q).
1627 Proof.
1628 intros. split.
1629 * apply TausAndCong_Implies_Reduction.
1630 * apply Reduction_Implies_TausAndCong.
1631 Qed.
```

A.2 Well Defined

With De Brujin indices, we introduced a Well-Defined abstraction ("closed terms") :

```
Inductive Well_Defined_Data : nat -> Data -> Prop :=
| bvar_is_defined_up_to_k: forall k x, (x < k) -> Well_Defined_Data k (bvar x)
| cst_is_always_defined : forall k v, Well_Defined_Data k (cst v).

Inductive Well_Defined_Condition : nat -> Equation Data -> Prop :=
| tt_is_WD : forall k, Well_Defined_Condition k tt
| ff_is_WD : forall k, Well_Defined_Condition k ff
| Inequality_is_WD : forall k x y, Well_Defined_Data k x -> Well_Defined_Data k y -> Well_Defined_Condition k (x ⩽ y)
| Or_is_WD : forall k e e', Well_Defined_Condition k e -> Well_Defined_Condition k e' -> Well_Defined_Condition k (e v e')
| Not_is_WD : forall k e, Well_Defined_Condition k e -> Well_Defined_Condition k (non e).

Inductive Well_Defined_Input_in : nat -> proc -> Prop :=
| WD_par : forall k p1 p2, Well_Defined_Input_in k p1 -> Well_Defined_Input_in k p2
                -> Well_Defined_Input_in k (p1 ‖ p2)
| WD_var : forall k i, Well_Defined_Input_in k (pr_var i)
| WD_rec : forall k x p1, Well_Defined_Input_in k p1 -> Well_Defined_Input_in k (rec x • p1)
| WD_if_then_else : forall k p1 p2 C, Well_Defined_Condition k C -> Well_Defined_Input_in k p1
                    -> Well_Defined_Input_in k p2
                        -> Well_Defined_Input_in k (If C Then p1 Else p2)
| WD_deadlock : forall k, Well_Defined_Input_in k (δ)
| WD_success : forall k, Well_Defined_Input_in k (①)
| WD_nil : forall k, Well_Defined_Input_in k (0)
| WD_input : forall k c p, Well_Defined_Input_in (S k) p
                  -> Well_Defined_Input_in k (c ? x • p)
| WD_output : forall k c v p, Well_Defined_Data k v -> Well_Defined_Input_in k p
                  -> Well_Defined_Input_in k (c ! v • p)
| WD_tau : forall k p,  Well_Defined_Input_in k p -> Well_Defined_Input_in k (t • p)
| WD_choice : forall k p1 p2,  Well_Defined_Input_in k (g p1) ->  Well_Defined_Input_in k (g p2)
              ->  Well_Defined_Input_in k (p1 + p2)
| WD_seq : forall k p1 p2,  Well_Defined_Input_in k p1 ->  Well_Defined_Input_in k p2
              ->  Well_Defined_Input_in k (p1 ; p2).
.............
.............
.............|
Inductive Well_Defined_Input_in_MultiSet : nat -> gmultiset TypeOfActions -> Prop :=
| WD_gmset_empty : forall k, Well_Defined_Input_in_MultiSet k ∅
| WD_gmset_base : forall k M d c, Well_Defined_Input_in_MultiSet k M -> Well_Defined_Data k d ->
            Well_Defined_Input_in_MultiSet k (M ⊎ {[+ c ⋉ d +]}).

Inductive Well_Defined_Input_in_State : nat -> States -> Prop :=
| WD_state_base : forall k M P, Well_Defined_Input_in_MultiSet k M -> Well_Defined_Input_in k P ->
            Well_Defined_Input_in_State k (⟨ M, P ⟩).
```

Indeed we can produce $bvar$ that are not linked, like in :

$$c\,!\,bvar\ 0 \bullet ⓪$$

And our objective for programming language are closed terms.

And we verified that the LTS and the STS we produced for States propagate the closed terms.

```
Lemma STS_Respects_WD : forall S S', Well_Defined_Input_in_State 0 S -> sts S S' -> Well_Defined_Input_in_State 0 S'.
Proof.
intros. revert H. rename H0 into Reduction. dependent induction Reduction.
* intros. dependent destruction H.
  dependent destruction H0. dependent destruction H0_.
  constructor.
  - eapply Well_Def_Subset with (M ⊎ {[+ c ⋉ v +]}).  assumption. multiset_solver.
  - assert (Well_Defined_Input_in_MultiSet 0 {[+ c ⋉ v +]}).
    eapply Well_Def_Subset with (M ⊎ {[+ c ⋉ v +]}). assumption. multiset_solver.
    eapply Well_Def_Singleton in H0.
    eapply Well_Def_Data_Is_a_value in H0.  destruct H0. subst.
    eapply ForSTS. assumption.
* intros. dependent destruction H. constructor.
  - constructor. assumption. dependent destruction H0. dependent destruction H0_. assumption.
  - dependent destruction H0. dependent destruction H0_. assumption.
* intros. dependent destruction H. dependent destruction H0. dependent destruction H0_. constructor; assumption.
* intros. dependent destruction H. constructor. assumption. apply RecursionOverReduction_is_WD. assumption.
* intros. dependent destruction H0. dependent destruction H1. constructor; assumption.
* intros. dependent destruction H0. dependent destruction H1. constructor; assumption.
* intros. dependent destruction H. dependent destruction H0. assert (Well_Defined_Input_in_State 0 ｛ M1, P1 ｝).
  constructor; assumption. apply IHReduction in H0. dependent destruction H0.
  constructor. assumption. constructor; assumption.
* intros. dependent destruction H. dependent destruction H0. dependent destruction H0_.
  assert (Well_Defined_Input_in_State 0 ｛ M1, P1 ｝).
  constructor; assumption. apply IHReduction in H0. dependent destruction H0.
  constructor. assumption. constructor; assumption.
* intros. dependent destruction H0. dependent destruction H1. dependent destruction H1_.
  constructor; assumption.
* intros. apply CongruenceM_Respects_WD with S2'. apply IHReduction. eapply CongruenceM_Respects_WD with S1.
  assumption. assumption. assumption.
Qed.

Inductive Well_Defined_Action: (Act TypeOfActions) -> Prop :=
| ActionOuput_with_value_is_always_defined : forall c v, Well_Defined_Action (ActOut  (c ⋉ (cst v)))
| ActionInput_with_value_is_always_defined : forall c v, Well_Defined_Action (ActIn  (c ⋉ (cst v)))
| Tau_is_always_defined : Well_Defined_Action (τ).

Lemma LTS_Respects_WD : forall S S' α, Well_Defined_Input_in_State 0 S -> Well_Defined_Action α -> ltsM S α S'
            ->  Well_Defined_Input_in_State 0 S'.
Proof.
intros. dependent induction H1.
- dependent destruction H.
  assert (Well_Defined_Input_in 0 q). eapply (PreLTS_Respects_WD p q (ActIn (c ⋉ v))); assumption.
  constructor; assumption.
- dependent destruction H. constructor.
  eapply Well_Def_Subset with (M ⊎ {[+ c ⋉ v +]}). assumption. multiset_solver.
  assumption.
- constructor.
  * dependent destruction H. assumption.
  * dependent destruction H. eapply (PreLTS_Respects_WD p q τ); assumption.
- dependent destruction H. constructor.
  * constructor. assumption. eapply Output_are_good in H0. instantiate (1 := v) in H0.
    destruct H0. subst. constructor. exact H2.
  * eapply (PreLTS_Respects_WD p q (ActOut (c ⋉ v))).
    assumption. eapply Output_are_good in H0. instantiate (1 := v) in H0. destruct H0. subst. econstructor.
    eassumption. exact H2.
- dependent destruction H1_. dependent destruction H1_0. dependent destruction H.
  constructor.
  * eapply Well_Def_Subset with (M2 ⊎ {[+ c ⋉ v +]}). assumption. multiset_solver.
  * eapply PreLTS_Respects_WD. exact H0.
    assert (Well_Defined_Data 0 v). eapply Well_Def_Singleton. eapply Well_Def_Subset.
    instantiate (1 := (M2 ⊎ {[+ c ⋉ v +]})). assumption.
    instantiate (1 := c). multiset_solver.
    instantiate (1 := ActExt (ActIn (c ⋉ v))). eapply Well_Def_Data_Is_a_value in H3.
    destruct H3. subst. constructor. assumption.
Qed.
```

A.3 LTS classes

Our main objective was to build a start to analyse the Must Pre-order Characterization from [27] with a enough expressive language.

So we proved that $(\mathcal{S}, \mathcal{A}, \xrightarrow{\alpha})$ was an instance of each classes from [27].

```
#[global] Program Instance VACCS_Label : Label TypeOfActions.
  (* {| label_eqdec := TypeOfActions_dec ;
     label_countable := TypeOfActions_countable
  |}. *)

#[global] Program Instance VACCS_Lts : Lts States TypeOfActions.
Next Obligation. intros. exact (ltsM X X0 X1). Defined. (* lts_step x ℓ y  := ltsM x ℓ y *)
Next Obligation. intros. exact (ltsM_dec a α b). Defined. (* lts_state_eqdec := proc_dec *)
Next Obligation. intros. exact (outputs_of_State X). Defined. (* lts_outputs S := outputs_of_State S *)
Next Obligation. intros. apply (outputsM_of_spec1 p1 x p2). apply H. Defined. (* lts_outputsM_spec1 p1 x p2 := outputs_of_spec1 p1 x p2 *)
Next Obligation. intros. apply (outputs_of_spec2 p1 x). assumption. Defined. (* lts_outputs_spec2 p1 x := outputs_of_spec2 p1 x *)
Next Obligation. intros. exact (states_stable X X0). Defined. (* lts_stable p α := states_stable p α *)
Next Obligation. intros. exact (states_stable_dec p α). Defined. (* lts_stable_decidable p α := states_stable_dec p α *)
Next Obligation. intros p [[a|a]|]; unfold VACCS_Lts_obligation_6 in *; unfold VACCS_Lts_obligation_1 in *;
        intro hs;eapply gset_nempty_ex in hs as (r & l); eapply ltsM_set_spec0 in l;
        exists r; assumption. Defined.
Next Obligation.
        intros p [[a|a]|]; intros (q & mem); intro eq; eapply ltsM_set_spec1 in mem; set_solver.
Qed.

#[global] Program Instance VACCS_LtsEq : LtsEq States TypeOfActions.
Next Obligation. exact cgr_states. Defined. (* eq_rel x y  := cgr x y *)
Next Obligation. apply cgr_states_refl. Qed. (* eq_rel_refl p := cgr_refl p *)
Next Obligation. apply cgr_states_symm. Qed. (* eq_symm p q := cgr_symm p q *)
Next Obligation. eapply cgr_states_trans. Qed. (* eq_trans x y z:= cgr_trans x y z *)
Next Obligation. eapply CongruenceStates_Respects_Transition. Qed. (* eq_spec p q α := CongruenceStates_Respects_Transition p q α *)

#[global] Program Instance VACCS_LtsOba : LtsOba States TypeOfActions.
Next Obligation. exact moutputs_of_State. Defined. (* lts_oba_mo p := moutputs_of p  *)
Next Obligation.
    intros. simpl. unfold VACCS_LtsOba_obligation_1. unfold VACCS_Lts_obligation_3. unfold outputs_of_State.
    now rewrite gmultiset_elem_of_dom.
  Qed.
Next Obligation.
    intros. unfold VACCS_LtsOba_obligation_1. simpl. unfold outputs_of_State.
    now eapply mo_spec.
  Qed.
Next Obligation. exact OBA_with_FB_First_Axiom. Qed. (* lts_oba_output_commutativity p q r a α := OBA_with_FB_First_Axiom p q r a α *)
Next Obligation. exact OBA_with_FB_Second_Axiom. Qed. (* lts_oba_output_confluence p q1 q2 a μ := OBA_with_FB_Second_Axiom p q1 q2 a μ *)
Next Obligation. exact OBA_with_FB_Fifth_Axiom. Qed. (* lts_oba_output_tau p q1 q2 a := OBA_with_FB_Fifth_Axiom p q1 q2 a *)
Next Obligation. exact OBA_with_FB_Third_Axiom. Qed. (* lts_oba_output_deter p1 p2 p3 a := OBA_with_FB_Third_Axiom p1 p2 p3 a *)
Next Obligation. exact ExtraAxiom. Qed. (* lts_oba_output_deter_inv p1 p2 q1 q2 a := ExtraAxiom p1 p2 q1 q2 a *)

#[global] Program Instance VACCS_LtsObaFB : LtsObaFB States TypeOfActions :=
  {| lts_oba_fb_feedback p1 p2 p3 a := OBA_with_FB_Fourth_Axiom p1 p2 p3 a |}.
```

Bibliography

[1] Peter Selinger (1997) *First-Order Axioms for Asynchrony*, Article, University of Pennsylvania.

[2] Robin Milner (1980) *A Calculus of Communicating Systems*, Book, Springer-Verlag - Berlin Heidelberg GmbH

[3] Davide Sangiorgi, David Walker (1980) *The π-calculus, A Theory of Mobile Processes*, Book, University of Cambridge

[4] Robin Milner (1999) *Communicating and mobile systems : the π-calculus*, Book, University of Cambridge

[5] Robin Milner (1991) *The Polyadic π-Calculus: a Tutorial*, Article, University of Edinburgh

[6] Shuqin Huang, Yongzhi Cao, Hanpin Wang, Wanling Qu (2012) *Value-passing CCS with noisy channels*, Article, University of Edinburgh

[7] Peter Selinger (1997) *Categorical Structure of Asynchrony*, Article, University of Michigan.

[8] Davide Sangiorgi (2011) *An Introduction to Bisimulation and Coinduction*, Book, University of Bologna

[9] Damien Pous (2017) *Coinduction All the Way Up*, Article, New York, United States.

[10] Andrew D. Gordon (1995) *A Tutorial on Co-induction and Functional Programming*, Article, University of Cambridge.

[11] Arthur Charguéraud (2012) *The Locally Nameless Representation*, Article, INRIA Rocquencourt.

[12] Matthew Hennessy (2007) *A Distributed Pi-Calculus*, Book, University of Cambridge.

[13] Matthew Hennessy, Anna Ingólfsdóttir (1993) *Communicating Processes with Value-passing and Assignments*, Book, University of Sussex.

[14] Rocco De Nicola, Matthew Hennessy (2007) *CCS without τ's*, Article, CNR - Pisa , University of Sussex.

[15] Robin Milner, Joachim Parrow, David Walker (1992) *A Calculus of Mobile Processes, II*, Article, University of Edinburgh, Swedish Institute of Computer Science - Kista, University of Warwick.

[16] Conor McBride, James McKinna (2004) *Functional Pearl: I am not a Number—I am a Free Variable*, Article, University of Durham, University of St Andrews.

[17] Naoki Kobayashi, Benjamin C. Pierce, David N. Turner (1999) *Linearity and the Pi-Calculus*, Article, The University of Tokyo, University of Pennsylvania.

[18] Anna Ingólfsdóttir (1994) *Late and early semantics coincide for testing*, Article, Aalborg University.

[19] Luca Padovani (2014) *Deadlock and lock freedom in the linear π-calculus*, Article, Università di Torino.

[20] Daniel Hirschkof (2019) *A brief survey of the theory of the Pi-calculus*, Article, CNRS-INRIA-ENS LYON.

[21] Johannes Aldert Bergstra, Jan Willem Klop (1984) *Process Algebra for Synchronous Communication*, Article, Center for Mathematics and Computer Science - Amsterdam.

[22] Rocco De Nicola (2013) *A gentle introduction to Process Algebras*, Article, IMT - Institute for Advanced Studies Lucca.

[23] Charles Antony Richard Hoare (1985-2022) *Communicating Sequential Processes*, Book.

[24] Hubert Garavel (2015) *Revisiting sequential composition in process calculi*, Article, Saarland University.

[25] Paolo Baldan, Filippo Bonchi, Fabio Gadducci, Giacoma Valentina Monreale (2012) *Concurrency Can't Be Observed, Asynchronously*, Article, University of Padova, Université de Lyon, Universita di Pisa.

[26] Clément Aubert (2020) *Quelle est la notion correcte de congruence structurelle pour une algèbre de processus?*, Article, Augusta University.

[27] Giovanni Bernardi, Ilaria Castellani, Paul Laforgue, Léo Stefanesco (2024) *Constructive characterisations of the must-preorder for asynchrony*, Article, Université Paris Cité, INRIA - Université Côte d'Azur, Nomadic Labs, MPI-SWS.

[28] Michele Boreale, Rocco De Nicola, Rosario Pugliese (2002) *Trace and Testing Equivalence on Asynchronous Processes*, Dipartimento di Sistemi e Informatica, Universita di Firenze.

[29] James H. Morris (1969) *Lambda-calculus models of programming languages. Ph. D. Dissertation.*

[30] Leslie Lamport (1977) *Proving the Correctness of Multiprocess Programs.*

[31] Bowen Alpern, Fred B. Schneider (1985) *Defining Liveness.*

[32] Rocco De Nicola, Matthew Hennessy (1984) *Testing Equivalences for Processes.*

[33] Ilaria Castellani, Matthew Hennessy. (1998) *Testing Theories for Asynchronous Languages.*

[34] Giovani Bernardi, Paul Laforgue, Léo Stefanesco (2024) *Machine-checked testing theory for asynchronous systems.*