A process calculus with clocks and priorities

Luigi Liquori, Michael Mendler, Claude Stolze

A process calculus with clocks and priorities

Luigi Liquori
Centre Inria de l'Université Côte d'Azur

Michael Mendler
University of Bamberg

Claude Stolze
University of Bamberg

Abstract

We define CCS^{spt}, a process algebra extending Milner's CCS with clocks and priorities on actions up-to clocks. It is designed to capture multi-clock synchronous programs in a compositional way, defining the scheduling of the processes through priorities. The novel features of CCS^{spt} allow us to model shared memory with reaction to absence of specific concurrent actions, in order to prevent race conditions and ensure determinacy. We present both a term rewriting system and a labelled transition system for CCS^{spt}, we prove the equivalence between these two presentations (called Harmony by Sangiorgi and Walker), and show some examples to highlight the usefulness of our system in practical cases.

2012 ACM Subject Classification Software and its engineering → Synchronization; Theory of computation → Process calculi

Keywords and phrases Timed process algebras, Synchronous programming

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Concurrency has become ubiquitous in modern softwares. However, designing concurrent programs is still error-prone because race conditions may easily appear while being difficult to detect and the resulting program ends up having an unpredictable behaviour and is difficult, if not impossible, to debug. One way to tackle the problem is using synchronous programming (SP), which started with Statecharts [20] and has generated languages such as Signal [18], Lustre [19], Esterel [5], Quartz [36], Zelus [10], SCCharts [38], Lingua Franca [24], and so on. In summary, synchronous programming impose a global synchronisation barrier (called the *clock*) for all processes, and impose determinacy between two clock actions (called a cycle, or an instant). Compilers for synchronous programs find a scheduler that enforce a write-before-read policy inside a cycle, e.g. reading a variable is done after it has finished being updated. Take for instance this Esterel program, which runs for a single cycle:

```
output O;
signal counter : unsigned init 0 combine + in
emit O if (?counter = 2) || emit ?counter <= 1 || emit ?counter <= 1
```

There are three concurrent processes in this program, each one using a counter, whose initial value is 0. Every number written to the counter is added to its current value, because of `combine +`. The first process emits a signal `O` iff the counter equals 2, while the two other processes write 1 to the counter, so the final value of the counter is 2. This looks like a classical case of read/write race condition, but, in Esterel, the behaviour of this program is determinate: the counter is incremented twice, then, as there is no further update for it, the first process reads its value, which is 2, and emits `O`. This example highlights a specific feature of synchronous programming: it is able to detect, inside a cycle, the *absence* of

further updates for a variable[1]. Only then it allows reading its value. This is quite tricky to do because a variable can be updated an unknown number of times each cycle (even zero times), independently of the number of threads. If the program cannot be correctly scheduled, it is either rejected by the compiler, or rejected at run-time, depending on the specific synchronous language and/or compiler you are using.

Extending these concepts to general objects is an interesting challenge. One proposal is to add scheduling policies to objects [1, 37]. The policy will give priorities between the methods of the object, and a method can be called if there is a guaranteed absence of further call of methods with higher priorities in the current cycle. Signals can then be encoded as simple objects with a setter and a getter, the setter having priority over the getter. We can also, for instance, implement a counter object, with two methods for incrementing and decrementing it, a method saying whether the counter is 0, and a getter method. Those last two methods would have the lowest priority, and the first two would have the highest priority, thus allowing incrementation and decrementation to happen in any order (because these operations commute), and when there is an absence of these method calls in the rest of the current cycle, we can call the other two methods.

Process algebras, like CCS [28], CSP [22], or π-calculus [29, 30], have been devised as a model of concurrency where a system and its environment continuously interact to mutually control each other's behaviour through synchronisation and communication actions. In CCS, the interaction of concurrent processes $P \,|\, Q$ arises from the rendezvous synchronisation of an action α of P and an associated co-action $\overline{\alpha}$ from Q, generating a silent action τ, also called a *reduction*.

This paper introduces an extension of CCS, denoted $\mathsf{CCS}^{\mathsf{spt}}$, that brings together the concepts of *priorities* [12, 34, 33] and *multiple-clocks* [4, 14, 7, 32] that have previously been studied independently. We thus obtain an adequate compositional setting for SP exploiting standard techniques from process algebra. This is novel since many different customised semantics (e.g., [26, 11, 5, 2, 1] for single-clocks, and [17, 27, 7] for multi-clocks) have been developed for SP over the years, few of which fit into the frameworks of classical process algebra. These semantics, even the most recent [1], give a different specification for shared memory and programs, while process algebras use a unified approach for both. In the other direction, none of the extensions of CCS by priorities [12, 34, 33] or clocks [4, 14, 32] have systematically investigated determinacy for shared memory scenarios. We choose CCS here, but similar ideas might be possible for other process algebras like ATP [31], CSP or some synchronous variants of Milner's π-calculus [3], just to mention a few. We find CCS most plausible as a point of departure since the concept of priorities is already well established and the combination of asynchronous scheduling and communication via local rendezvous synchronisation makes the problem of ensuring determinacy more prominent.

We call our process algebra 'SynPaTick', denoted $\mathsf{CCS}^{\mathsf{spt}}$. At the technical level, we make the following specific contributions in this paper:

- We use broadcast actions (clocks) as global synchronisation barriers to schedule processes using priorities in a more flexible way than any of the earlier approaches. While all classical extensions of CCS by priorities [12, 34, 33] schedule the processes by their immediate initial actions, our system takes into account all actions potentially executable up to the next clock barrier, giving us a scheduling of the threads which is constructive. It was surprising for us to discover that such a natural modification suffices to obtain

[1] In the Esterel terminology, shared variables are called *signals*.

a process-algebraic model for shared-memory multi-threading, such as required for the constructive semantics of Esterel [5].
- We present two operational semantics: a Term Rewriting System (TRS), which give a reduction relation and therefore a standard way to compute our programs, and a Labelled Transition System (LTS), which details interactions between processes.
- We show that these two semantics are in harmony.

The harmony result for the π-calculus has been formulated by Sangiorgi and Walker [35]. From a metalanguage point of view, we found it inspiring to quote a personal communication by Honsell [23]:

> *"The significance of a Harmony result arises in that two conceptually different approaches to semantics, namely the initial and the final one, are proved to correspond to each other, and hence they essentially capture the same semantic equivalence. Transition systems permit to define inductively, bottom-up, congruences, i.e. they yield naturally compositional equivalences. Labelled transition systems are conceptually dual to them. They give rise to co-inductive equivalences, i.e. greatest fixed points, and take a global top-down view".*

2 A Process Algebra with Clocks and Priorities

The syntax and the TRS and LTS operational semantics of $\mathsf{CCS}^{\mathsf{spt}}$ are defined in this section.

Notation and Syntax Let $\mathcal{A}$ be a countably infinite set of channel names with $a, b \ldots$ ranging over $\mathcal{A}$. As in CCS, the set $\overline{\mathcal{A}}$ is a disjoint set of co-names with elements denoted by $\overline{a}, \overline{b} \ldots$ in bijection with $\mathcal{A}$. The overbar operator switches between $\mathcal{A}$ and $\overline{\mathcal{A}}$, i.e., $\overline{\overline{a}} = a$ for all $\overline{a} \in \overline{\mathcal{A}}$. We will refer to the $a \in \mathcal{R} \stackrel{def}{=} \mathcal{A} \cup \overline{\mathcal{A}}$ as rendezvous actions (or rendezvous labels). Let $\mathcal{C}$ be a countably infinite set of broadcast clock names, disjoint from $\mathcal{R}$ and let $\sigma, \rho \ldots$ range over $\mathcal{C}$. Vectors of channels and clocks can be denoted by $\vec{a}$ and $\vec{\sigma}$, respectively. Every clock acts as its own co-name, $\overline{\sigma} = \sigma$ and so each $\sigma \in \mathcal{C}$ is also considered as a clock action (or clock label). Let $\mathcal{L} = \mathcal{R} \cup \mathcal{C}$ be the set of rendezvous and clock labels and let ℓ range over $\mathcal{L}$, while L, H range over subsets of $\mathcal{L}$. We write $\overline{L}$ for the set $\{\overline{\ell} \mid \ell \in L\}$. Let $\mathcal{L}^\tau = \mathcal{L} \cup \{\tau\}$ be the set of all actions obtained by adjoining the silent action $\tau \notin \mathcal{L}$ and let α range over $\mathcal{L}^\tau$. We extend the overbar operator to $\mathcal{L}^\tau$ by defining $\overline{\tau} = \tau$. All symbols can appear indexed.

$\mathsf{CCS}^{\mathsf{spt}}$ terms $\mathcal{T} = \mathcal{P} \cup \mathcal{S}$ come in two mutually recursive syntactic forms, processes $\mathcal{P}$ and threads $\mathcal{S}$. Let $\mathcal{I} \subset \mathcal{P}$ be a set of process names, and let $\mathsf{p}, \mathsf{q} \ldots$ range over $\mathcal{I}$. The process terms $P, Q \in \mathcal{P}$ and the threads terms $M, N \in \mathcal{S}$ are defined by the following abstract syntax:

P, Q	$::=$	M	thread	M, N	$::=$	0_C	inactive process
	$\mid$	$\mathsf{p}(\vec{a}, \vec{\sigma})$	process name, $\mathsf{p} \in \mathcal{I}$		$\mid$	$\alpha{:}L.P$	prefix, $\alpha \in \mathcal{L}^\tau, L \subseteq \mathcal{L}$
	$\mid$	$P \,\vert\, Q$	parallel composition		$\mid$	$M + N$	sum
	$\mid$	$P \setminus A$	restriction, $A \subseteq \mathcal{A}$.				
	$\mid$	$P \,/\, C$	hiding, $C \subseteq \mathcal{C}$				

We assume that in every restriction $P \setminus A$, we have $A \subseteq \mathcal{A}$, that is, A contains only rendezvous actions, and in every hiding $P \,/\, C$ we have $C \subseteq \mathcal{C}$. These scoping operators act as name binders which introduce local scopes. Intuitively:

- 0_C is the inactive process for the set of clocks C, it can do nothing;
- $\alpha{:}L.P$ is the process that may become P after performing the action α, but the set $L \subseteq \mathcal{L}$ contains all the actions taking precedence over it, that is, if a concurrent process may

synchronise on an action in L before a clock tick, then this process is blocked. Note that always $\tau \notin L$. This is because silent actions cannot synchronise and thus cannot be used to block a prefix. An action $\alpha \in \mathcal{R}$ denotes a CCS-style rendezvous (or handshake) action. An action $\alpha \in \mathcal{C}$ denotes a CSP-style broadcast (or clock) action, that shall synchronise with all the surrounding processes in the scope where this clock has been declared;

- $M + N$ is the sum of M and N, that is, it progress either as M or N;
- $P \,|\, Q$ is the parallel composition of P and Q. As in other process calculi, they can proceed independently and also interact through synchronisation of actions. However, in CCS$^{\mathsf{spt}}$, they can also indirectly interact by preventing the other process from proceeding, thanks to the priority system;
- $P \setminus A$ and P / C denote respectively action restriction and clock hiding; they make local the rendezvous action in A, and respectively the broadcasted clocks in C;
- p process name refers to a predefined process with arguments.

A useful abbreviation is to omit the blocking sets if they are empty or drop the continuation process if it is $\mathsf{0}_C$ and C is unimportant. For instance, we will typically write $a.P$ instead of $a{:}\{\}.P$ and $a{:}L$ rather than $a{:}L.\mathsf{0}_C$. Also, it is useful to drop the braces for the blocking set writing $a{:}b.P$ instead of $a{:}\{b\}.P$. We assume that the unary operators of prefix, restriction and hiding take precedence over the binary operators and we write $P \setminus c \,|\, R / \sigma$ instead of $(P \setminus c) \,|\, (R / \sigma)$. We consider the restriction and hiding operators to be binders: processes are identified modulo α-conversion. The set of free labels (or free variables) $\mathcal{L}(P) \subseteq \mathcal{L}$ of a process P is defined as follows:

$$
\begin{array}{rclcrcl}
\mathcal{L}(\mathsf{0}_C) & = & C & \qquad & \mathcal{L}(P \setminus A) & = & \mathcal{L}(P) - (A \cup \overline{A}) \\
\mathcal{L}(\ell{:}L.P) & = & \{\ell\} \cup \mathcal{L}(P) \text{ if } \ell \in \mathcal{L} & & \mathcal{L}(\mathsf{p}(\vec{a}, \vec{\sigma})) & = & \{\vec{a}, \vec{\sigma}\} \\
\mathcal{L}(\tau{:}L.P) & = & \mathcal{L}(P) & & \mathcal{L}(P \,|\, Q) & = & \mathcal{L}(P) \cup \mathcal{L}(Q) \\
\mathcal{L}(P / C) & = & \mathcal{L}(P) - C & & \mathcal{L}(M + N) & = & \mathcal{L}(M) \cup \mathcal{L}(N)
\end{array}
$$

The clocks of a process P, denoted $\mathsf{clocks}(P)$, is the set defined as $\mathcal{L}(P) \cap \mathcal{C}$. We want processes who preserve their set of clocks during their execution. The two critical operations are sums and prefixes, so every thread M as well as its direct sub-expressions must have the same set of clocks. More formally, we say a process is well-defined if all its sub-expressions of the shape $\alpha{:}L.P$ verify $\mathsf{clocks}(\alpha{:}L.P) = \mathsf{clocks}(P)$ and all its sub-expressions of the shape $M_1 + M_2$ verify $\mathsf{clocks}(M_1) = \mathsf{clocks}(M_2)$. As counter-examples, the processes $\rho.\mathsf{0}_\sigma$ and $\sigma.\mathsf{0}_\sigma + \rho.\mathsf{0}_\rho$ are not well-defined, while $\rho.\mathsf{0}_{\sigma,\rho}$ and $\sigma.\mathsf{0}_\sigma \,|\, \rho.\mathsf{0}_\rho$ are well-defined.

From now on, we will only consider well-defined processes. A process P has to synchronise on every *tick* of clocks in $\mathsf{clocks}(P)$, and only those clocks. It corresponds to what Hoare [22] calls the alphabet of a CSP process. Note the special role of process $\mathsf{0}_C$: more than simply doing nothing, it also does not synchronise on any clock in C, thus preventing these clocks from ticking.

Process names It is tricky to properly define process names in CCS, because process names can be intuitively seen as functions whose arguments are their free names, so accommodating α-conversion and argument passing can be cumbersome. Milner's [28] original presentation uses an explicit relabelling function and a process of the form $P[f]$. This paper use the solution by Phillips [33] to explicitly pass a vector of channel names $\vec{a}$ and one of clocks $\vec{\sigma}$ to process names, which are then substituted during the unfolding of their definition. We may remove arguments from process name, hence writing p instead of $\mathsf{p}(\vec{a}, \vec{\sigma})$ if the arguments are either useless or easy to recover.

Initial actions The set of initial actions of a thread M, noted $\mathsf{iA}(M) \subseteq \mathcal{L}^\tau$, is defined by

$$\mathsf{iA}(0_C) = \{\} \qquad \mathsf{iA}(\alpha{:}L.P) = \{\alpha\} \qquad \mathsf{iA}(M+N) = \mathsf{iA}(M) \cup \mathsf{iA}(N)$$

The originality of our priority mechanism is that we have priorities of actions up-to clocks, so we define the set of initial actions up-to a set of clocks $\mathsf{iA}^*_C(P)$ of a process relative to a set of clocks $C \subseteq \mathcal{C}$. The presence of process names makes the definition a bit difficult, so we first define the set of initial actions up-to a set of clocks $C \subseteq \mathcal{C}$ and a number of unfolding of process names $n \in \mathbb{N}$. We define the set $\mathsf{iA}^n_C(P) \subseteq \mathcal{L}$ by the following recursive rules:

$$\begin{array}{rcll}
\mathsf{iA}^n_C(0_{C'}) & = & \{\} & \\
\mathsf{iA}^n_C(\alpha{:}L.P) & = & \begin{cases} \{\alpha\} & \text{if } \alpha \in C \\ \{\alpha\} \cup \mathsf{iA}^n_C(P) & \text{otherwise} \end{cases} \\
\mathsf{iA}^n_C(P \,/\, C') & = & \mathsf{iA}^n_{C-C'}(P) - C' & \\
\mathsf{iA}^n_C(P \setminus A) & = & \mathsf{iA}^n_C(P) - (A \cup \overline{A}) &
\end{array}
\qquad
\begin{array}{rcll}
\mathsf{iA}^0_C(\mathsf{p}) & = & \{\} & \\
\mathsf{iA}^{n+1}_C(\mathsf{p}) & = & \mathsf{iA}^n_C(P) & \text{if } \mathsf{p} \stackrel{def}{=} P \\
\mathsf{iA}^n_C(P \,|\, Q) & = & \mathsf{iA}^n_C(P) \cup \mathsf{iA}^n_C(Q) & \\
\mathsf{iA}^n_C(M+N) & = & \mathsf{iA}^n_C(M) \cup \mathsf{iA}^n_C(N) &
\end{array}$$

We define $\mathsf{iA}^*_C(P) = \bigcup_{n\in\mathbb{N}} \mathsf{iA}^n_C(P)$, as $\mathsf{iA}^n_C(P)$ is monotonic for n (i.e. $\mathsf{iA}^n_C(P) \subseteq \mathsf{iA}^{n+1}_C(P)$). One tricky case arises when we hide local clocks: local clock names may shadow global ones. For instance, the rules give us that $\mathsf{iA}^*_{\{\sigma\}}((\sigma.a)\,/\,\sigma) = \{a\}$, because we are looking for the actions that can be done up to the *global* clock σ, which is not the same as the *local* clock σ.

Operational Semantics Following the seminal book of Sangiorgi and Walker [35], $\mathsf{CCS}^{\mathsf{spt}}$ is presented by means of both a TRS and a LTS. The two relations are respectively called reductions and transitions, and are noted by $P \to Q$ and $P \xrightarrow[B]{\alpha}_\iota Q$ where $\alpha \in \mathcal{L}^\tau$ is the action that labels the transition, $B \subseteq 2^{\mathcal{C}} \times 2^{\mathcal{L}}$ is the *blocking* relation, and $\iota \in 2^{\mathcal{C}} \longrightarrow 2^{\mathcal{L}^\tau}$ is the *prediction* function. As usual, the TRS explains the internal behaviour of processes, whereas the labelled transition system explains how processes interact with themselves and their surroundings. These two systems are in harmony in the sense that the unblockable τ-transitions of a process are the same as its reductions.

Blocking Relation and Prediction Functions The scheduling control for both the semantic relations is achieved by blocking relations B and prediction functions ι associated with an action or a process.

Intuitively, each pair $(C, L) \in B$ of a blocking relation is a blocking constraint that gives us the actions L which must not be able to synchronise until one of the clocks of C has been executed. An action $a \in \mathcal{L}^\tau$ is considered a competing action within the prediction horizon given by C, if there exists $(C, L) \in B$ with $a \in L$. For instance, if $B = \{(\{\sigma\}, \{a\}), (\{\rho_1, \rho_2\}, \{\overline{b}\})\}$, then we require that a must not happen up to σ, and $\overline{b}$ must not happen up to either ρ_1 or ρ_2. A prediction function ι gives us, for a set of clocks C, a set of competing actions which *may* happen before any of the clocks in C is executed.

Blocking relation and prediction functions work together to determine the enabledness of actions. If ι is a prediction function and B a blocking relation, then define the property "ι *eschews* B" by

$$\mathsf{eschews}(\iota, B) \stackrel{def}{=} \forall (C, L) \in B,\ \iota(C) \cap \overline{L} = \{\}$$

This notion extends the one by Phillips [33] and expresses that the process or action with prediction function ι will not block another process from performing an action whose blocking relation is B. While in [33] the prediction (called "*offerings*" of a process) and blocking are simply sets of actions, in our multi-clock setting they are scoped by sets of clocks.

$$
\begin{array}{rcl@{\qquad}rcl}
M_1 + (M_2 + M_3) & \equiv & (M_1 + M_2) + M_3 & P_1 \,|\, (P_2 \,|\, P_3) & \equiv & (P_1 \,|\, P_2) \,|\, P_3 \\
M_1 + M_2 & \equiv & M_2 + M_1 & P_1 \,|\, P_2 & \equiv & P_2 \,|\, P_1 \\
M + \mathsf{0}_{\mathsf{clocks}(M)} & \equiv & M & P \,|\, \mathsf{0}_{\{\}} & \equiv & P \\
\mathsf{0}_{C_1} / C_2 & \equiv & \mathsf{0}_{C_1 - C_2} & P \setminus A_1 \setminus A_2 & \equiv & P \setminus (A_1 \cup A_2) \\
P / C_1 / C_2 & \equiv & P / (C_1 \cup C_2) & P \setminus A / C & \equiv & P / C \setminus A
\end{array}
$$

$$
\begin{array}{rcll}
P \setminus A & \equiv & P & \text{if } \mathcal{L}(P) \cap (A \cup \overline{A}) = \{\} \\
P \,|\, (Q \setminus A) & \equiv & (P \,|\, Q) \setminus A & \text{if } \mathcal{L}(P) \cap (A \cup \overline{A}) = \{\} \\
P / C & \equiv & P & \text{if } \mathcal{L}(P) \cap C = \{\} \\
P \,|\, (Q / C) & \equiv & (P \,|\, Q) / C & \text{if } \mathcal{L}(P) \cap C = \{\} \\
\mathsf{p}(\vec{a}, \vec{\sigma}) & \equiv & P & \text{if } \mathsf{p}(\vec{a}, \vec{\sigma}) \stackrel{def}{=} P
\end{array}
$$

Figure 1 Structural congruence for $\mathsf{CCS}^{\mathsf{spt}}$

Operations on Blocking Relations and Prediction Functions The blocking relation B is a property of a transition and constructed in a compositional fashion by the structural rules of our labelled semantics defined below. The basic relations are singletons of form $B = \{(C, L)\}$ capturing the blocking set L of a single action along with the clocks C of the thread executing the action. In the synchronisation of an actions we combine their blocking relations by set union $B_1 \cup B_2$. The restriction $B \setminus A$ of a blocking relation simply removes the actions $A \cup \overline{A}$ from each entry in B, i.e., $B \setminus A = \{(C, L - (A \cup \overline{A})) \mid (C, L) \in B\}$. The operation of hiding clocks in a blocking relation is $B / C = \{(C' - C, L - C) \mid (C', L) \in B\}$. Here we remove the local clocks both from the clocks and the blocking set of the transition.

The prediction functions are used in two ways, as properties of processes and of transitions. Our basic prediction functions are the empty prediction function $\iota^{\{\}}$ which is defined as $\iota^{\{\}}(C) = \{\}$ for all C, and the prediction function for a process P, noted $\mathsf{iA}^*_-(P)$, which returns $\mathsf{iA}^*_C(P)$ for all C. As operations on prediction functions we have point-wise union written as a summation: $\iota_1 + \iota_2$ is defined as $\forall C \subseteq \mathcal{C},\ (\iota_1 + \iota_2)(C) = \iota_1(C) \cup \iota_2(C)$. The extension $\iota \cup L$ of a prediction function ι with a set of actions $L \subseteq \mathcal{L}^\tau$, is defined as $\forall C \subseteq \mathcal{C},\ (\iota \cup L)(C) = \iota(C) \cup L$. Further, we want to restrict a prediction ι by removing a set of actions $A \subseteq \mathcal{A}$ pointwise, written $\iota \setminus A$ and defined as $\forall C \subseteq \mathcal{C},\ (\iota \setminus A)(C) = \iota(C) - (A \cup \overline{A})$. Finally, we can hide a set of clocks $C \subseteq \mathcal{C}$ in a prediction: ι / C is defined as $\forall C' \subseteq \mathcal{C},\ (\iota / C)(C') = \iota(C' - C) - C$. Notice that we obtain ι / C by subtracting the hidden (local) clocks C both from the argument and the result of the prediction function. By taking $C' - C$ in the argument of ι we make the prediction function skip the local clocks C, taking only the visible clocks as the synchronisation horizon. Since the local clocks C are skipped and returned by $\iota(C' - C)$ we must remove them from the output prediction, too.

Structural congruence The relation $\equiv$ on terms is the smallest congruence induced by the rules in Fig. 1 together with usual rules of equational reasoning such as reflexivity, symmetry, transitivity, and congruence. It is easy to see that, if $P \equiv Q$ and P is well-defined, then Q is well-defined, and $\mathcal{L}(P) = \mathcal{L}(Q)$. In contrast to usual process algebras, notice that $\mathsf{0}_{\{\}}$ is the neutral element for the parallel composition (the inactive process with no clocks), while for sums of threads whose set of clocks is C, the neutral element for the sum is $\mathsf{0}_C$.

2.1 Term Rewriting System

Processes are equivalent up to a *canonical* form $(M_1 \mid \cdots \mid M_n)\,/\,C \setminus A$, where each thread M_i is itself either $\mathsf{0}_{C_i}$ for some C_i, or a sum of processes prefixed by an action. The TRS for $\mathsf{CCS}^{\mathsf{spt}}$ is presented in Fig. 2. Let's start by explaining the (*Tau*), (*RdV*), and (*Clock*) rules. For each reduction, a group of threads synchronise on some action, and proceed. For the (*Tau*) rule, only one thread does an action. For the (*RdV*) rule, two threads synchronise by doing dual rendezvous actions. For the (*Clock*) rule, all the threads using a common local clock synchronise by doing the clock action. However, these actions can be prevented by the surrounding context. For each acting thread M, its action $\alpha{:}L$ is decorated with a set of local labels L containing all actions that take precedence over α. The novel feature of our calculus, the $\mathsf{eschews}$ predicate, ensures the precedences are respected. Each thread is concerned with what could happen until some point (that we call the *horizon*), generally until one of its clocks ticks. So each thread checks that the actions in L cannot happen up to some horizon.

The notion of horizon is not trivial. For non-cooperating threads, we can assume the priority system should check what they can do up to some clocks, but what about cooperating threads? Can they block each other? As a simple case, let us consider $M = a{:}b.\mathsf{0}_\sigma$ and $N = \overline{a}{:}b.\mathsf{0}_\rho + \tau{:}\overline{a}.\overline{b}.\mathsf{0}_\rho$. N can do a τ-action, then a $\overline{b}$-action, thus, if it does not cooperate with M, it blocks the a-action of M. However, if M and N synchronise on a, then the branch $\tau{:}\overline{a}.\overline{b}.\mathsf{0}_\rho$ in N will never be taken, and it would be silly to block M for an action that will never happen. However, if we consider $N' = \overline{a}{:}b.\mathsf{0}_\rho + \overline{b}.\mathsf{0}_\rho$, can M and N' synchronise on a? In this case, they cannot, because $\overline{b}$ in N' is an immediate action, which intuitively means that the immediate actions of a thread are relevant. As a consequence, the prediction function is the iA^* function for the non-cooperating threads, augmented with the iA set of the cooperating threads (minus the action being performed). For instance, in the first side condition of the rule (*RdV*), it is, for the thread $a{:}L.P + M$, the prediction function $\iota = \mathsf{iA}^*_{-}(R) \cup (\mathsf{iA}(N) - \{\overline{a}\})$, and the $\mathsf{eschews}$ predicate ensures that $\iota(\mathsf{clocks}(P)) \cap \overline{L} = \{\}$, putting together the clocks up to which we have to check for precedences $\mathsf{clocks}(P)$, the blocking constraints L, and the prediction function ι. Thus $\iota(\mathsf{clocks}(P))$ is the formalization of the horizon.

The priority system allows us to check for absence: if $a{:}L.P$ can proceed, we are sure that none of the actions in $\overline{L}$ can happen up to $\mathsf{clocks}(P)$. Using τ actions, as in the (*Tau*) rule, allows us to check for absence without doing anything else: for instance, $a.P + \tau{:}a.Q$ intuitively means "if a is feasible in the current cycle, then do a then P, else do Q".

For the remaining rules: rule (*Congr*) is standard and needs no comments, while rules (Par_{lts}), ($Restr_{\mathsf{lts}}$), and ($Hide_{\mathsf{lts}}$) define syntactic closure.

2.2 Labelled Transition System

The LTS for $\mathsf{CCS}^{\mathsf{spt}}$ is presented as the least relation of form $P \xrightarrow[B]{\alpha}_{\iota} Q$ closed under the rules of Fig. 3. This transition expresses that "P performs the action α with the blocking relation B and the prediction function ι, and becomes Q". Also, we will henceforth talk about B as the *blocking* and about ι as the *prediction* of the transition. Sometimes, we write $P \xrightarrow{\alpha} Q$ to state that there is a transition for some B and ι. In the following, we take a closer look at the adornments B and ι and their role in the scheduling of a transition $\xrightarrow[B]{\alpha}_{\iota}$.

Each element $(C, L) \in B$ is a blocking constraint from a thread participating in the transition, that runs in the scope of clocks C and only participates in the transition provided there cannot be a synchronisation on any of the competing actions L in the clock horizon

$$\frac{\mathsf{eschews}(\mathsf{iA}^*_-(R) \cup (\mathsf{iA}(N) - \{\overline{a}\}), \{(\mathsf{clocks}(P), L)\}) \quad \mathsf{eschews}(\mathsf{iA}^*_-(R) \cup (\mathsf{iA}(M) - \{a\}), \{(\mathsf{clocks}(Q), L')\}) \quad L \cup L' \subseteq C \cup A \cup \overline{A}}{((a{:}L.P + M) \mid (\overline{a}{:}L'.Q + N) \mid R) / C \setminus A \to (P \mid Q \mid R) / C \setminus A} \; (RdV)$$

$$\frac{\bigcup_i L_i \subseteq C \cup A \cup \overline{A} \quad \sigma \in C \quad \sigma \notin \mathsf{clocks}(R) \quad \forall i,\ \mathsf{eschews}(\mathsf{iA}^*_-(R) \cup (\bigcup_{j \neq i} \mathsf{iA}(M_j) - \{\sigma\}), \{(\mathsf{clocks}(P_i), L_i)\})}{((\sigma{:}L_1.P_1 + M_1) \mid \cdots \mid (\sigma{:}L_n.P_n + M_n) \mid R) / C \setminus A \to (P_1 \mid \cdots \mid P_n \mid R) / C \setminus A} \; (Clock)$$

$$\frac{P \equiv P' \to Q' \equiv Q}{P \to Q} \; (Congr) \qquad \frac{\mathsf{eschews}(\mathsf{iA}^*_-(R), \{(\mathsf{clocks}(P), L)\}) \quad L \subseteq C \cup A \cup \overline{A}}{((\tau{:}L.P + M) \mid R) / C \setminus A \to (P \mid R) / C \setminus A} \; (Tau)$$

$$\frac{P \to Q}{P \mid R \to Q \mid R} \; (Par_{\mathsf{trs}}) \qquad \frac{P \to Q}{P \setminus A \to Q \setminus A} \; (Restr_{\mathsf{trs}}) \qquad \frac{P \to Q}{P / C \to Q / C} \; (Hide_{\mathsf{trs}})$$

Figure 2 Term Rewriting System for $\mathsf{CCS}^{\mathsf{spt}}$

$$\frac{P_1 \xrightarrow[B_1]{\ell} \iota_1 P_1' \quad P_2 \xrightarrow[B_2]{\overline{\ell}} \iota_2 P_2' \quad \forall i \in \{1,2\},\ \mathsf{eschews}(\iota_i, B_{3-i})}{P_1 \mid P_2 \xrightarrow[B_1 \cup B_2]{\ell \mid \overline{\ell}} \iota_1 + \iota_2 P_1' \mid P_2'} \; (Com)$$

$$\frac{\mathsf{p}(\vec{a}, \vec{\sigma}) \stackrel{def}{=} P \quad P \xrightarrow[B]{\alpha} \iota Q}{\mathsf{p}(\vec{a}, \vec{\sigma}) \xrightarrow[B]{\alpha} \iota Q} \; (Con) \qquad \frac{}{\alpha{:}L.P \xrightarrow[\{(\mathsf{clocks}(P), L)\}]{\alpha} \iota^{\{\}} P} \; (Act) \qquad \frac{M_1 \xrightarrow[B]{\alpha} \iota P}{M_1 + M_2 \xrightarrow[B]{\alpha} \iota \cup (\mathsf{iA}(M_2) - \{\alpha\})} P \; (Sum)$$

$$\frac{\mathsf{eschews}(\mathsf{iA}^*_-(Q), B) \quad P \xrightarrow[B]{\alpha} \iota P' \quad \alpha \notin \mathsf{clocks}(Q)}{P \mid Q \xrightarrow[B]{\alpha} \iota + \mathsf{iA}^*_-(Q) P' \mid Q} \; (Par_{\mathsf{lts}}) \qquad \frac{P \xrightarrow[B]{\alpha} \iota Q \quad \alpha \notin A \cup \overline{A}}{P \setminus A \xrightarrow[B \setminus A]{\alpha} \iota \setminus A Q \setminus A} \; (Restr_{\mathsf{lts}}) \qquad \frac{P \xrightarrow[B]{\alpha} \iota Q}{P / C \xrightarrow[B / C]{\alpha / C} \iota / C Q / C} \; (Hide_{\mathsf{lts}})$$

Figure 3 Labelled Transition System for $\mathsf{CCS}^{\mathsf{spt}}$

given by C. The clock horizon consists of all actions possible taken before any of the clocks in C is executed. If $\alpha \in \mathcal{R}$ is a rendezvous action then B contains a single blocking constraint, if $\alpha = \tau$ the relation B contains two constraints, one from each rendezvous partner, and if $\alpha = \sigma$ then B contains one constraint from each thread synchronising in the clock transition.

The prediction ι takes a set of clocks C and returns the set of actions that are in competition to $\xrightarrow[B]{\alpha} \iota$ within the clock horizon C. An action is in competition if it is either an immediate alternative choice of any of the threads partaking in $\xrightarrow[B]{\alpha} \iota$, or if it is an action potentially offered by a thread that runs concurrently with the threads partaking in α. Let us now look at the rules in Fig. 3. The behaviour of threads is captured by the rules (*Act*) and (*Sum*). General processes P are essentially trees of threads combined in parallel with lexical scoping of local clocks and local actions. The concurrent execution is handled by the rules (*Com*) and (Par_{lts}), the scoping of labels is handled by ($Hide_{\mathsf{lts}}$) and ($Restr_{\mathsf{lts}}$).

- (*Act*) The execution of a prefix $\alpha{:}L.P$ publishes the empty prediction function $\iota^{\{\}}$, because it does not have any alternative choices or concurrency that would compete. The blocking

$\{(\mathsf{clocks}(P), L)\}$ lifts the blocking set L of the prefix to a blocking constraint in the scope of the thread's clock. Note, by well-formedness, $\mathsf{clocks}(\alpha{:}L.P) = \mathsf{clocks}(P)$.

- (*Sum*) The iterated application of the summation rules permits us to select any prefix $\alpha_n{:}L_n.P_n$ of a thread $M = \sum_i \alpha_i{:}L_i.P_i$, generating a transition $\xrightarrow[\iota]{\alpha}_B$ of M to P_n, with $B = \{(\mathsf{clocks}(P_n), L_n)\}$ and $\iota = \iota^{\{\}} \cup \{\alpha_i \mid i \neq n, \alpha_i \neq \alpha_n\}$. The blocking B is that of the prefix. Note that by well-formedness $\mathsf{clocks}(P_n) = \mathsf{clocks}(M)$. The prediction ι contains the initial actions α_i for all other choices $i \neq n$ offered by thread M as competitors to α_n. Note that α_n does not count as a competitor for itself, whence we subtract it.
- (*Com*) The parallel composition rule implements simultaneously the synchronisation of rendezvous actions and clocks: For rendezvous actions $a \in \mathcal{R}$ we set $a \,|\, \overline{a} = \tau$. Since τ is a non-synchronising (silent) action, for which (*Com*) is not applicable any more, this closes the two-party synchronisation. For contrast, a clock $\sigma \in \mathcal{C}$ always synchronises with itself $\overline{\sigma} = \sigma$ and since $\sigma \,|\, \overline{\sigma} = \sigma$ it remains open for more participants, thus generating a multi-party synchronisation. In either case, the construction of blocking and prediction for the $\ell \,|\, \overline{\ell}$ transition is the same: We take the union $B_1 \cup B_2$ of the blocking constraints and the point-wise union $\iota_1 + \iota_2$ of the predictions from the two participating transitions ℓ and $\overline{\ell}$. In addition we must check the enabledness of the synchronisation $\ell \,|\, \overline{\ell}$ cross-wise: the prediction ι_i of P_i must eschew the blocking B_{3-i} for $i \in \{1, 2\}$.
- (Par_{lts}) These rules describe the asynchronous case where one process P in a parallel composition $P \,|\, Q$ executes a step unsynchronised with Q. The blocking B is inherited from the transition α of P. This transition is only enabled as an asynchronous step of P in concurrency with Q, if the prediction $\mathsf{iA}^*_{\_}(Q)$ of Q eschews the blocking B and if the action α is not a clock of Q. For if $\alpha \in \mathsf{clocks}(Q)$ then P cannot proceed alone, it must lock-step synchronise with Q via rule (*Com*). Since the actions predicted by $\mathsf{iA}^*_{\_}(Q)$ are in (concurrent) competition to α, we add them to the prediction ι.
- ($Hide_{\mathsf{lts}}$) The clock hiding $P \,/\, C$ makes the fresh local clocks C available in P. Every action of P is an action of $P \,/\, C$, except if P executes a local clock $\sigma \in C$. Then, just like in CSP for hidden broadcast actions, this action is exposed as a silent τ to the outside. In this way, the hiding operator closes the broadcast synchronisation on the clocks. To express this, the rule uses the auxiliary notation $\alpha \,/\, C$ which is defined as $\alpha \,/\, C \stackrel{def}{=} \alpha$ if $\alpha \notin C$ and $\alpha \,/\, C \stackrel{def}{=} \tau$ if $\alpha \in C$. Since the clocks C are local to P, we eliminate them from the blocking and prediction, using the auxiliary operations $B \,/\, C$ and $\iota \,/\, C$.
- ($Restr_{\mathsf{lts}}$) The restriction $P \setminus A$ of rendezvous actions A is as in CCS. It prunes away all local transitions with labels from $A \cup \overline{A}$, because these are no longer available for synchronisation outside. Naturally then, we must remove these labels from the blocking and predictions as well, which is done by the operations $B \setminus A$ and $\iota \setminus A$.
- (*Con*) This is the standard unfolding rule for constant definitions, for our new transition relation that includes blocking and predictions.

3 Examples

The following examples illustrate the expressivity of $\mathsf{CCS}^{\mathsf{spt}}$. The focus is on how the transition labels, blocking relations and prediction functions implement precedences, and in some cases we can achieve program determinism.

▸ **Example 1** (Counter). Let us recall the counter example from the introduction.

```
output O;
signal counter : unsigned init 0 combine + in
```

```
emit 0 if (?counter = 2) || emit ?counter <= 1 || emit ?counter <= 1
```

$\mathsf{CCS}^{\mathsf{spt}}$ does not allow value-passing, but we can still implement this example. We will not use any clocks. The counter requires three actions: inc, is2, and isnot2. The inc action takes precedence over the two others. We will define an infinite set of process names $\mathsf{counter}_i$ for each integer i, with

$$\mathsf{counter}_i \stackrel{def}{=} \begin{cases} \mathrm{inc}.\mathsf{counter}_{i+1} + \overline{\mathrm{isnot2}}{:}\mathrm{inc}.0_{\{\}} & \text{if } i \neq 2 \\ \mathrm{inc}.\mathsf{counter}_{i+1} + \overline{\mathrm{is2}}{:}\mathrm{inc}.0_{\{\}} & \text{if } i = 2 \end{cases}$$

The main program becomes:

$$P \stackrel{def}{=} (\mathrm{is2}.\overline{o}.0_{\{\}} + \mathrm{isnot2}.0_{\{\}}) \,|\, \overline{\mathrm{inc}}.0_{\{\}} \,|\, \overline{\mathrm{inc}}.0_{\{\}}$$

That is, P emits o if the counter is 2, and it also increments the counter twice. Note that P does not declare any priority, they are taken care of by the counter process. The process $P \,|\, \mathsf{counter}_0$ cannot reduce, because there is no guarantee an external process will not increment the counter again. However, we can reduce $(P \,|\, \mathsf{counter}_0) \setminus \mathrm{inc}$. We can synchronise one of the $\overline{\mathrm{inc}}$ action from P with the inc action from $\mathsf{counter}_0$, and we obtain a *unique* TRS reduction path (modulo syntactic congruence):

$$\begin{aligned} (P \,|\, \mathsf{counter}_0) \setminus \{\mathrm{inc}\} &\to ((\mathrm{is2}.\overline{o}.0_{\{\}} + \mathrm{isnot2}.0_{\{\}}) \,|\, \overline{\mathrm{inc}}.0_{\{\}} \,|\, \mathsf{counter}_1) \setminus \{\mathrm{inc}\} \\ &\to ((\mathrm{is2}.\overline{o}.0_{\{\}} + \mathrm{isnot2}.0_{\{\}}) \,|\, \mathsf{counter}_2) \setminus \{\mathrm{inc}\} \\ &\to \overline{o}.0_{\{\}}. \end{aligned}$$

In the $(P \,|\, \mathsf{counter}_0) \setminus \mathrm{inc}$ process, we cannot synchronise the $\overline{\mathrm{isnot2}}$ action from $\mathsf{counter}_0$ with the isnot2 action from P, because the side condition

$$\mathsf{eschews}(\mathsf{iA}^*_{\_}(\overline{\mathrm{inc}}.0_{\{\}} \,|\, \overline{\mathrm{inc}}.0_{\{\}}) \cup (\mathsf{iA}(\mathrm{isnot2}.0_{\{\}}) - \{\overline{\mathrm{is2}}\}), \{(\mathsf{clocks}(\mathsf{counter}_0), \{\mathrm{inc}\})\})$$

simplifies as $\{\overline{\mathrm{inc}}, \mathrm{isnot2}\} \cap \overline{\{\mathrm{inc}\}} = \{\}$, which is false, which means that the inc action blocks the isnot2 action. What about the same synchronisation transition path in the LTS? We have

$$P \xrightarrow[(\{\},\{\})]{\mathrm{isnot2}}_{\iota} \overline{\mathrm{inc}} \,|\, \overline{\mathrm{inc}},$$

with $\iota(C) = \{\overline{\mathrm{inc}}, \mathrm{is2}\}$ for any C, which intuitively means that P can perform the isnot2 action without any blocking, while announcing its $\overline{\mathrm{inc}}$ and is2 actions may block other processes. Meanwhile we have

$$\mathsf{counter}_0 \xrightarrow[(\{\},\{\mathrm{inc}\})]{\overline{\mathrm{isnot2}}}_{\iota'} 0_{\{\}},$$

with $\iota'(C) = \{\mathrm{inc}\}$ for any C, which means that $\mathsf{counter}_0$ can perform the $\overline{\mathrm{isnot2}}$ action as long as it is not blocked by $\overline{\mathrm{inc}}$, and it could also perform inc. Those two derivations cannot synchronise, because the ι function from the first transition blocks the second transition.

▸ **Example 2** (Processes with different clocks). Consider the following processes:

$$1(\sigma) \stackrel{def}{=} \sigma.1 \qquad \mathsf{getStr} \stackrel{def}{=} \mathrm{enter}.\overline{\mathrm{str}}.1(\sigma) + \mathrm{key}{:}\mathrm{enter}.\sigma.\mathsf{getStr} \qquad \mathsf{P} \stackrel{def}{=} \mathrm{str}.1(\rho) + \rho{:}\mathrm{str}.\mathsf{P}$$

Here, the process $1(\sigma)$ (resp. $1(\rho)$) does σ actions (resp. ρ actions) *ad libidum*. If we consider, for instance $\rho.(\mathsf{getStr} \,|\, \mathsf{P})$, then we have a process that does a ρ action, then P waits for getStr to perform a $\overline{\mathrm{str}}$ action. Because of the priority in ρ:str, any clock action ρ is prevented because getStr may perform a $\overline{\mathrm{str}}$ action before the clock ρ.

$$
\begin{array}{lcl}
\mathsf{ABRO} & \stackrel{def}{=} & (\mathsf{A} \,|\, \mathsf{B} \,|\, \mathsf{R} \,|\, \mathsf{O}) \setminus \{s, k\} \\
1 & \stackrel{def}{=} & \sigma.1 \\
\mathsf{A} & \stackrel{def}{=} & k{:}k.1 + a{:}\{k,a\}.\overline{s}{:}\overline{s}.1 + \sigma{:}\{k,a\}.\mathsf{A} \\
\mathsf{B} & \stackrel{def}{=} & k{:}k.1 + b{:}\{k,b\}.\overline{s}{:}\overline{s}.1 + \sigma{:}\{k,b\}.\mathsf{B} \\
\mathsf{R} & \stackrel{def}{=} & r{:}r.\mathsf{R}' + \tau{:}r.\sigma.\mathsf{R} \\
\mathsf{R}' & \stackrel{def}{=} & \overline{k}.\mathsf{R}' + \sigma{:}\overline{k}.\mathsf{ABRO} \\
\mathsf{O} & \stackrel{def}{=} & k{:}k.1 + s{:}k.\mathsf{O}' + \sigma{:}\{k,s\}.\mathsf{O} \\
\mathsf{O}' & \stackrel{def}{=} & k{:}k.1 + s{:}k.\mathsf{O}'' + \sigma{:}\{k,s\}.\mathsf{O}' \\
\mathsf{O}'' & \stackrel{def}{=} & s{:}k.\mathsf{O}'' + \overline{o}{:}\{\overline{o},s\}.1
\end{array}
$$

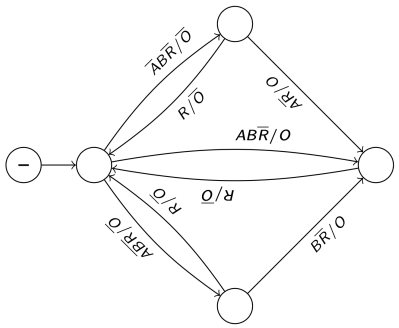

Figure 4 ABRO as a CCS^{spt} process and a Mealy machine

▸ **Example 3** (Changing the frequency of a clock)**.** In hardware, clock frequency refers to the rate at which a clock progresses, indicating how quickly it counts time. Consider the following processes:

$$
\begin{array}{rcl}
\mathsf{P}(x,\sigma) & \stackrel{def}{=} & \mathsf{P}_1(x,\sigma,\rho) \,/\, \rho \\
\mathsf{P}_1(x,\sigma,\rho) & \stackrel{def}{=} & \rho{:}\{x,\sigma\}.(\sigma{:}x.\mathsf{P}_1(x,\sigma,\rho) + x{:}x.\sigma{:}x.\mathsf{P}_2(x,\sigma,\rho)) + x{:}x.\sigma{:}x.\mathsf{P}_2(x,\sigma,\rho) \\
\mathsf{P}_2(x,\sigma,\rho) & \stackrel{def}{=} & \rho{:}\{x,\sigma\}.(\rho{:}\{x,\sigma\}.(\sigma{:}x.\mathsf{P}_2(x,\sigma,\rho) + x{:}x.\sigma{:}x.\mathsf{P}_1(x,\sigma,\rho)) + \\
 & & x{:}x.\sigma{:}x.\mathsf{P}_1(x,\sigma,\rho)) + x{:}x.\sigma{:}x.\mathsf{P}_1(x,\sigma,\rho).
\end{array}
$$

If we consider that the action $\overline{x}$ is never offered, then $\mathsf{P}_1(x,\sigma,\rho) \,/\, \rho$ triggers the internal clock ρ once between each clock σ, whereas $\mathsf{P}_2(x,\sigma,\rho) \,/\, \rho$ triggers ρ twice between each σ. The action x allows the process P to switch between $\mathsf{P}_1(x,\sigma,\rho) \,/\, \rho$ and $\mathsf{P}_2(x,\sigma,\rho) \,/\, \rho$.

▸ **Example 4** (ABRO)**.** The *Hello, World!* of the Esterel synchronous language [6] has the following behaviour: *"Emit the output o as soon as both inputs a and b have been received. Reset this behaviour each time the input r occurs, without emitting o in the reset cycle"* [16]. The synchronous behaviour of Esterel lies in the assumption that each transition of the Mealy machine summarises a single *macro-step* interaction of the machine with its environment. The transitions of parallel machines synchronise so that state changes proceed in lock-step. The simultaneous execution of these transitions that make up a macro-step consists of the sending and receiving of signals. The propagation of individual signals between machines is modeled in the *micro-step* operational semantics of Esterel.

```
module ABRO:
 input A, B, R;
 output O;
 loop
  await A || await B;
  emit O
 each R
end module
```

The behaviour of ABRO is expressed as a CCS^{spt} process and a Mealy machine in Fig. 4. In CCS^{spt} we represent the micro-steps as sequences of τ-transitions, whereas the macro-step is captured as a global clock synchronisation. Unfortunately, the Mealy machine formalism is *not scalable* because the number of states can grow *exponentially* as the number of signals grow, while in Esterel can be expressed in a linear number of code lines as seen on the right. The `loop P each R` construct of Esterel executes the behaviour of program `P` for an unbounded number of clock cycles, but restarts its body `P` when the signal `R` becomes present.

The reset `R` takes priority over the behaviour of `P` which is preempted at the moment that the reset occurs. Note that in Esterel each signal appears *exactly once*, as in the specification and unlike in the Mealy machine.

Other parts of Esterel also naturally arise from precedence-based scheduling. Esterel's `loop P each R` mechanism, for instance, can be coded using the static parallel composition operator that combines the process `P` to be aborted and reset with a *watchdog* process `R`. The latter waits for the reset signal from the environment in each clock cycle. The choice between the reset and continuing inside `P` is resolved by giving `R` higher priority. When the reset occurs, the watchdog sends *kill* signals to all threads running in `P` and waits for these to terminate. Only then the watchdog `R` restarts `P` from scratch. This is a form of precedence that can be naturally expressed in the blocking sets of $\mathsf{CCS}^{\mathsf{spt}}$.

In our ABRO encoding, we assume that the channels a, b, r, o model the signals `A`, `B`, `R`, `O`, and all have a unique sender and receiver. The main process ABRO creates local actions s and k, and executes $\mathsf{A} \mid \mathsf{B} \mid \mathsf{R} \mid \mathsf{O}$. Hence, $\mathsf{A} \mid \mathsf{B}$ encodes `await A || await B`. The channel s is used to signal the normal ending of processes A and B to O: in other words, it ensures `await A || await B` is executed *before* `emit O`. Once s has been received at least twice by O (which encodes `emit O`), it emits $\overline{o}$ and ends, unless it is killed at some moment by the action k. Finally, R controls the Esterel loop: it kills all the remaining processes when it receives r using the action $\overline{k}$ which has priority over everything else, and once it cannot send $\overline{k}$ anymore, it waits for σ and restart ABRO. However, because r has the highest priority, process R has to decide whether r is present or absent before the concurrent processes can proceed. The $\tau{:}r$ action in R is triggered iff action r is not possible during that cycle, thus unblocking the other processes. Every action that is supposed to synchronise with a unique process is self-blocking. It is the case for every action except $\overline{k}$, s, and σ. Having as many self-blocking actions as possible helps ensuring the *determinacy* of the process.

4 Harmony

In this section, we show that the TRS and the LTS are equivalent. An appendix contains the full proofs. This harmony result, proved in Lemma 9, builds a bridge between the *computational* viewpoint (silent τ transitions modeling β-reductions as in λ-calculus) with the *concurrency* viewpoint (visible $\mathcal{L}$ transition as synchronising actions). The existence of such bridges have been identified by Abramsky for the lazy λ-calculus and by Sangiorgi and Walker for the π-calculus [35]. Harmony results are rarely evident and non-trivial (see e.g., the Coq proof by Momigliano [13]).

First we prove that syntactic equivalence commutes with the transition system (Lemma 5), then we show a canonical representation of processes (modulo equivalence) according to the transition they can perform (Lemmas 6, 7, and 8), before proving the main Harmony Lemma 9.

▸ **Lemma 5** (Syntactic $\equiv$). *If $P \equiv P'$ and $P \xrightarrow[B]{\alpha}_{\iota} Q$, then, for some $Q \equiv Q'$, $P' \xrightarrow[B]{\alpha}_{\iota} Q'$.*

Proof. By induction on the structural congruence rules $\equiv$. ◂

Then, we show that processes able to perform a rendezvous action in the transition system can be rewritten into a canonical form.

▸ **Lemma 6** (Channel a). *If $P \xrightarrow[B]{a}_{\iota} P'$, then $P \equiv ((a{:}L.Q + M) \mid R) / C \setminus A$ with*

- *$P' \equiv (Q \mid R) / C \setminus A$, where blocking and predictions are*
- *$B = \{(\mathsf{clocks}(Q), L)\} / C \setminus A$ and $\iota = (\mathit{iA}^*_-(R) \cup (\mathit{iA}(M) - \{a\})) / C \setminus A$.*

Moreover, the unblocking condition $\mathsf{eschews}(\mathit{iA}^*_-(R), \{(\mathsf{clocks}(Q), L)\})$ is satisfied.

Proof. By induction on the rules of the LTS. ◂

In a similar way, processes able to perform a clock action in the transition system can be rewritten into a canonical form. One verifies that, up to hiding and restriction, the blocking of a clock transition is the union of the blocking of all clock prefixes participating in the clock. The prediction is the iA* prediction of the concurrent threads not participating in the clock together with the initial actions iA of all threads minus the clock itself.

▸ **Lemma 7** (Clock σ). *If* $P \xrightarrow[B]{\sigma}_{\iota} P'$, *then*
$P \equiv ((\sigma{:}L_1.Q_1 + M_1) \mid \cdots \mid (\sigma{:}L_n.Q_n + M_n) \mid R) \,/\, C \setminus A$ *with*
$P' \equiv (Q_1 \mid \cdots \mid Q_n \mid R) \,/\, C \setminus A$ *and* $\sigma \notin \mathsf{clocks}(R) \cup C$, *where*

- *the blocking set is* $B = \bigcup_{i \leqslant n} \{(\mathsf{clocks}(Q_i), L_i)\} \,/\, C \setminus A$ *and*
- *the prediction function is* $\iota = (\mathit{iA}^*_-(R) \cup \bigcup_{i \leqslant n} \mathit{iA}(M_i) {-} \{\sigma\}) \,/\, C \setminus A$, *and*
- *the transition is unblocked, i.e.,* $\forall i, j \leqslant n$ *with* $i \neq j$, $\mathsf{eschews}(\mathit{iA}^*_-(R), \{(\mathsf{clocks}(Q_i), L_i)\})$ *and* $\overline{L_i} \cap (\mathit{iA}(M_j) {-} \{\sigma\}) = \{\}$.

Proof. By induction on the rules of the LTS. ◂

Finally, we show that processes able to perform a τ action in the transition system can be rewritten into a canonical form, which look closely to what we have in the (*Tau*), (*RdV*) and (*Clock*) rules of the reduction system. However, note that for τ-transitions the prediction ι is uninteresting since it cannot synchronise any more.

▸ **Lemma 8** (Silent action τ). *If* $P \xrightarrow[B]{\tau}_{\iota} P'$, *then we have three cases:*

- *(*τ *action)* $P \equiv ((\tau{:}L.Q + M) \mid R) \,/\, C \setminus A$ *with* $P' \equiv (Q \mid R) \,/\, C \setminus A$, *and*
  - $B = \{(\mathsf{clocks}(Q), L)\} \,/\, C \setminus A$, *and*
  - *the reduction is unblocked, i.e.,* $\mathit{iA}^*_{\mathsf{clocks}(Q)}(R) \cap \overline{L} = \{\}$.
- *(Rendezvous synchronisation)* $P \equiv ((a{:}L_1.Q_1 + M_1) \mid (\overline{a}{:}L_2.Q_2 + M_2) \mid R) \,/\, C \setminus A$ *and* $P' \equiv (Q_1 \mid Q_2 \mid R) \,/\, C \setminus A$, *with*
  - $B = \{(\mathsf{clocks}(Q_1), L_1), (\mathsf{clocks}(Q_2), L_2)\} \,/\, C \setminus A$, *and*
  - *the reduction is unblocked, i.e., for* $i \in \{1, 2\}$, *we have* $\mathsf{eschews}(\mathit{iA}^*_-(R), \{(\mathsf{clocks}(Q_i), L_i)\})$ *and both* $\overline{L_1} \cap (\mathit{iA}(M_2) {-} \{\overline{a}\}) = \{\}$ *and* $\overline{L_2} \cap (\mathit{iA}(M_1) {-} \{a\}) = \{\}$.
- *(Clock synchronisation)* $P \equiv ((\sigma{:}L_1.Q_1 + M_1) \mid \cdots \mid (\sigma{:}L_n.Q_n + M_n) \mid R) \,/\, C \setminus A$ *and* $P' \equiv (Q_1 \mid \cdots \mid Q_n \mid R) \,/\, C \setminus A$, *with*
  - $B = \bigcup_i \{(\mathsf{clocks}(Q_i), L_i)\} \,/\, C \setminus A$, *and*
  - $\sigma \in C$ *and* $\sigma \notin \mathsf{clocks}(R)$, *and*
  - *the reduction is unblocked, i.e., for all* $i, j \leqslant n$, $i \neq j$: $\mathsf{eschews}(\mathit{iA}^*_-(R), \{(\mathsf{clocks}(Q_i), L_i)\})$ *and* $\overline{L_i} \cap (\mathit{iA}(M_j) {-} \{\sigma\}) = \{\}$.

Proof. By induction on the rules of the LTS using Lemma 6 and Lemma 7. ◂

Using Lemmas 5 and 8, we can show that, on the one hand, reductions correspond to unblockable τ-transitions, and that, on the other hand, unblockable τ-transitions correspond to reductions.

▸ **Lemma 9** (Harmony). $P \longrightarrow Q$ *iff* $P \xrightarrow[B]{\tau}_{\iota} Q'$ *and* $Q' \equiv Q$, *for some* ι, Q', *and* B *such that* $\forall (C, L) \in B$, $L = \{\}$.

Proof. The direction ($\Rightarrow$) proceeds by induction on the TRS rules. When the reduction is obtained by the (*Congr*) rule we use Lemma 5. For the other direction ($\Leftarrow$) we employ Lemma 8. ◂

5 Related Work

Related Work For our extension of Milner's CCS, to capture synchronous programming, priorities and clocks are instrumental. Our priority guards are different from those in previous work [12, 33, 34] where the blocking of synchronisations is made on the basis of the *immediate* initial actions iA. This earlier form of scheduling implements the idea that the scheduler receives control after each action of every thread and that after each action, the scheduling situation can change in arbitrary ways. Here in CCS$^{\mathsf{spt}}$, instead, we use the transitive closure iA^*_C of all *reachable* actions of P within a given clock horizon C. This corresponds to the idea that the scheduler operates in larger bursts of macro-steps that represent the logical ticks of a (multi-clock) synchronous model of computation. Our scheduling model based on look-ahead predictions iA^*_C is more conservative and enjoys an important monotonicity property: once an action is unblocked, it remains unblocked until any of the clocks from the horizon C ticks. After the barrier, the scheduling can again behave arbitrarily, until the next barrier. The change from *immediate enabling* based on iA to *up-to-clocks enabling* based on iA* is the cornerstone for modelling of the synchronous micro-macro-step abstraction of the constructive semantics of Esterel-style languages [5, 38]. This permits compositional coding of synchrony with reaction to absence as in the constructive semantics of Esterel. How the classical immediate enabling iA can be used to code synchronous Statecharts has been shown by [26] using a CCS-style algebra, dubbed SPL. But this only captures a weak form of reaction to absence and depends on special syntactic operators.

Priority guards in CCS introduce negative premises into the LTS. The general principle of LTS with negative premises has been studied by Bol and Groote [9]. The priority guards here and in Phillips are special, because they are not negations of the transition relation itself but they are defined independently via predicates iA and iA*. This avoids many of the complications of the general theory. The use of priorities for system specification is also fundamental in the BIP algebra of interactions by Bliudze and Sifakis [8] for modelling connectors rather than behaviours.

The second element borrowed from classical process algebra is the notion of clocks. While Andersen and Mendler's PMC [4] has multiple clocks but no priorities, in single-clock Hennessy and Regan's TPL [21], Nicollin and Sifakis ATP [31], Lüttgen *et al.* SPL [26] and Cleaveland *et al.* multi-clock CSA [14], the priorities are hardwired and express precedence only between a rendezvous action and a clock. Also, in contrast to the variants of timed CCS [21, 31, 14], CCS$^{\mathsf{spt}}$ provides a fully general priority scheme as in Phillips' CCS$^{\mathsf{Ph}}$ [33], without need for additional operators. We treat all actions on a par and do not need the timeout operator $\lfloor M \rfloor \sigma(Q)$ which in CCS$^{\mathsf{spt}}$ is simply $M + \sigma{:}\mathcal{L}.Q$. Moreover, all the cited extensions of CCS use immediate enabling iA like in CCS$^{\mathsf{Ph}}$, not to up-to-clocks iA* as in CCS$^{\mathsf{spt}}$, which is novel.

Expressivity of CCS$^{\mathsf{spt}}$ Firstly, due to blocking, CCS$^{\mathsf{spt}}$ is strictly more expressive than CCS. It is easy to see that there is no compositional encoding of CCS$^{\mathsf{spt}}$ into CCS. Consider the terms $\overline{a}{:}\overline{a}$ and $a{:}a$ which have one transition each, with action $\overline{a}$ and a, respectively: when composed in parallel, $\overline{a}{:}\overline{a} \,|\, a{:}a$ has only one possible transition, performing τ. However, there are no CCS processes P and Q which perform one transition each, and, put in parallel, only perform one transition, because parallel composition in CCS preserves the transitions of the composed processes. Secondly, unlike CCS and CCS$^{\mathsf{Ph}}$, there is no expansion lemma in CCS$^{\mathsf{spt}}$, i.e., processes cannot always be rewritten as a single thread. For instance, observe that $P = a \,|\, a$ cannot do a τ-transition when put in parallel with $\overline{a}{:}\overline{a}$. Yet, a single thread

M able to perform a in some transition $M \xrightarrow[B]{a}_{\iota} Q$ has the property that $\forall C \subseteq \mathcal{C},\ a \notin \iota(C)$. Therefore, $M \,|\, \overline{a}{:}\overline{a}$ can perform a τ-transition, and therefore P cannot be bisimilar to M. Thirdly, priorities in $\mathsf{CCS}^{\mathsf{spt}}$ do not seem as powerful as those in $\mathsf{CCS}^{\mathsf{Ph}}$. Define $a.P \oplus b.Q$ as an abbreviation of $(a{:}c.(P \,|\, \overline{c})) \,|\, (b{:}c.(Q \,|\, \overline{c})) \setminus c$. In Phillips' semantics, this sum $\oplus$ indeed acts like a non-deterministic free choice $a.P + b.Q$. In our semantics, $\oplus$ behaves deterministically, and we like it! It blocks, because of the strong predictions $\{a, \overline{c}\} \subseteq \mathsf{iA}^*(a{:}c.(P \,|\, \overline{c}))$ which can see the blocking action $\overline{c}$. Thus, we conjecture that summation $+$ cannot be coded in terms of prioritised prefixes and restricted parallel composition, unlike in $\mathsf{CCS}^{\mathsf{Ph}}$.

As indicated in our examples (Sec. 3), we can express synchronous programming with reaction to absence, or more generally (multi-clocked) shared-memory and multi-threading. But the exact characterisation of expressiveness of $\mathsf{CCS}^{\mathsf{spt}}$ we must leave as an open problem. We hope that this work may lead to a systematic study of the algebraic properties of (forms of) synchronous programming in the setting of classical process algebra and concurrent λ-calculi. Towards this end, it will be interesting scaling up our theory from CCS to the π-calculus, the higher-order process algebra par excellence, would not simply be a stylistic exercise. Challenges, such as causal dependencies introduced by scope extrusion, are significant and deserve careful analysis, as discussed in [15]. Event structures à la Winskel [39] could undoubtedly play a pivotal role in addressing these issues. More on the elementary side, we plan to expand our theory by standard instruments such as notions of bisimulation and associated algebraic axiomatisations. In view of applications to synchronous programming and concurrent λ-calculi, we are particularly interested in $\mathsf{CCS}^{\mathsf{spt}}$ as a setting for generalising determinacy and confluence (see [28][Chap.11]) to notions of coherence (see [25]).

Acknowledgements. The authors are grateful to Furio Honsell and Robert de Simone for their insightful comments and the careful reading of the paper.

References

1 J. Aguado, M. Mendler, M. Pouzet, P. S. Roop, and R. von Hanxleden. Deterministic concurrency: A clock-synchronised shared memory approach. In Proc. of ESOP, pages 86–113, 2018.

2 J. Aguado, M. Mendler, R. von Hanxleden, and I. Fuhrmann. Grounding synchronous deterministic concurrency in sequential programming. In Proc. of ESOP, LNCS 8410, pages 229–248. Springer, 2014.

3 R. M. Amadio. A synchronous pi-calculus. Inf. Comput., 205(9):1470–1490, 2007.

4 H. R. Andersen and M. Mendler. An asynchronous process algebra with multiple clocks. In D. Sannella, editor, In Proc. ESOP, pages 58–73, 1994.

5 G. Berry. The Constructive Semantics of Pure Esterel. Draft Book, 1999.

6 G. Berry. The Foundations of Esterel. In Proof, Language and Interaction: Essays in Honour of Robin Milner, pages 425–454. MIT Press, 2000.

7 G. Berry and E. Sentovich. Multiclock Esterel. In Proc. of CHARME, pages 110–125. Springer, 2001.

8 S. Bliudze and J. Sifakis. The algebra of connectors: structuring interaction in BIP. In Proc. of EMSOFT, pages 11–20. ACM, 2007.

9 R. N. Bol and J. F. Groote. The meaning of negative premises in transition system specifications. J. ACM, 43(5):863–914, 1996.

10 T. Bourke and M. Pouzet. Zélus: a synchronous language with ODEs. In Proc. of HSCC, pages 113–118. ACM, 2013.

11 F. Boussinot and R. De Simone. The SL synchronous language. IEEE Transactions on Software Engineering, 22(4):256–266, 1996.

12 J. Camilleri and G. Winskel. CCS with priority choice. Information and Computation, 116(1):26–37, 1995.

13 G. Cecilia and A. Momigliano. A Beluga formalization of the harmony lemma in the π-calculus. In Proc. of LFMTP, volume 404 of EPTCS, pages 1–17, 2024.

14 R. Cleaveland, G. Lüttgen, and M. Mendler. An algebraic theory of multiple clocks. In Proc. of CONCUR, number 1243 in LNCS, pages 166–180, 1997.

15 S. Crafa, D. Varacca, and N. Yoshida. Event structure semantics of parallel extrusion in the pi-calculus. In Proc. of FOSSACS, volume 7213 of LNCS, pages 225–239. Springer, 2012.

16 Esterel Technologies. The Esterel v7 reference manual version v7_30 – initial IEEE standardization proposal. Technical report, Esterel Technologies, 2005.

17 M. Gemünde, J. Brandt, and K. Schneider. Clock refinement in imperative synchronous languages. EURASIP J. Emb. Sys., 2013:3, 2013.

18 P. Le Guernic, T. Gautier, M. Le Borgne, and C. Le Maire. Programming real time applications with SIGNAL. In Proceedings of the IEEE, volume 79, pages 1321–1336. IEEE Press, 1991.

19 N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language LUSTRE. Proceedings of the IEEE, 79(9):1305–1320, 1991.

20 D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman. On the formal semantics of Statecharts. In Proc. of LICS, pages 54–64. IEEE Press, 1987.

21 M. Hennessy and T. Regan. A process algebra for timed system. Information and Computation, 117:221–239, 1995.

22 C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985. 256 pages.

23 F. Honsell, 2026. private correspondence.

24 S. Lin, Y. A. Manerkar, M. Lohstroh, E. Polgreen, S-J. Yu, C. Jerad, E. A. Lee, and S. A. Seshia. Towards building verifiable CPS using Lingua Franca. ACM Trans. Embed. Comput. Syst., 22(5s):155:1–155:24, 2023.

25 L. Liquori and M. Mendler. Strong priority and determinacy in timed CCS. hal-04367635, `https://arxiv.org/abs/2403.04618`, 2024.

26 G. Lüttgen, M. von der Beeck, and R. Cleaveland. Statecharts via process algebra. In Proc. of CONCUR, volume 1664 of LNCS, pages 399–414. Springer, 1999.

27 L. Mandel, C. Pasteur, and M. Pouzet. Time refinement in a functional synchronous language. In Proc. of PPDP, pages 169–180, 2013.

28 R. Milner. Communication and Concurrency. Prentice Hall, 1989.

29 R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. Inf. Comput., 100(1):1–40, 1992.

30 R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, II. Inf. Comput., 100(1):41–77, 1992.

31 X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: theory and application. Inf. Comput., 114(1):131–178, 1994.

32 B. Norton, G. Lüttgen, and M. Mendler. A compositional semantic theory for component-based synchronous design. In Proc. of CONCUR, pages 461–476. Springer LNCS 2761, 2003.

33 I. Phillips. CCS with priority guards. In K. G. Larsen and M. Nielsen, editors, Proc. of CONCUR, pages 305–320, 2001.

34 V. Natarajan R. Cleaveland, G. Lüttgen. Handbook of Process Agebra, chapter Priority in Process Algebra. Elsevier, 2001.

35 D. Sangiorgi and D. Walker. The π-calculus: a theory of mobile processes. Cambridge university press, 2003.

36 K. Schneider. The synchronous programming language Quartz. Internal Report 375, Department of Computer Science, University of Kaiserslautern, 2009.

37 A. Schulz-Rosengarten, S. Smyth, and M. Mendler. Towards object-oriented modeling in SCCharts. In Proc. of FDL, 2019.

38 R. von Hanxleden, B. Duderstadt, C. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, and O. O'Brien. SCCharts: Sequentially Constructive Statecharts for safety-critical applications. In Proc. of PLDI. ACM, 2014.

39 G. Winskel. Event structures. In Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, Germany, 8-19 September 1986, volume 255 of Lecture Notes in Computer Science, pages 325–392. Springer, 1986.

A Proofs

Proof of Lemma 5 (Syntactic $\equiv$).
By induction on the structural congruence rules.

- reflexive case: trivial;
- transitive case: easy;
- symmetric case: we cannot prove this case directly. Instead, we will prove the property holds for every base case and their symmetrical counterparts;
- case $P = M_1 + (M_2 + M_3) \equiv (M_1 + M_2) + M_3 = P'$: By disjunction on whether M_1, M_2, or M_3 is fired. The cases are symmetrical, so let's say M_1 is fired, and $M_1 \xrightarrow[B]{\alpha}_{\iota'} M_1'$. Then we can check that $\forall C \in \mathcal{C}$, $\iota(C) = \iota'(C) \cup iA(M_2 + M_3)$, and $Q = Q' = M_1'$, and we can conclude;
- case $M_1 + M_2 \equiv M_2 + M_1$: similarly;
- case $P = M + 0_{\mathsf{clocks}(M)} \equiv M = P'$: We have that $M \xrightarrow[B]{\alpha}_{\iota} M'$ for some M', and $Q = Q' = M'$;
- case $P = P_1 \,|\, (P_2 \,|\, P_3) \equiv (P_1 \,|\, P_2) \,|\, P_3 = P'$: By disjunction on which sub-expressions are fired. There are 7 different cases. In each case, we have processes $P_i \xrightarrow[B_i]{\alpha_i}_{\iota_i} P_i'$, and the final blocking relation $B = \bigcup_i B_i$, and the final prediction function $\forall C \in \mathcal{C}$, $\iota(C) = \bigcup_i \iota_i(C) \cup (\bigcup_{(j \text{ s.t. } P_j \text{ is not fired})} \mathsf{iA}^*_C(P_j))$. We can also check that

$$\forall (C, L) \in B_i,\ \left(\bigcup_{j \neq i} \iota'(C) \quad \cup \bigcup_{j \text{ s.t. } P_j \text{ is not fired}} \mathsf{iA}^*_C(P_j) \right) \quad \cap \quad \overline{L} \quad = \{\}$$

  therefore all the side-conditions hold;
- case $P_1 \,|\, P_2 \equiv P_2 \,|\, P_1$: similarly;
- case $P = P' \,|\, 0_{\{\}} \equiv P'$: there is only one case to consider: $P' \,|\, 0_{\{\}} \xrightarrow[B]{\alpha}_{\iota} Q' \,|\, 0_{\{\}}$ iff $P' \xrightarrow[B]{\alpha}_{\iota} Q'$;
- case $P' \setminus A \equiv P'$ if $\mathcal{L}(P') \cap (A \cup \overline{A}) = \{\}$: similarly;
- case $P = R \setminus A_1 \setminus A_2 \equiv R \setminus (A_1 \cup A_2)$: easy;
- case $P = P_1 \,|\, (Q_1 \setminus A) \equiv (P_1 \,|\, Q_1) \setminus A = P'$ if $\mathcal{L}(P_1) \cap (A \cup \overline{A}) = \{\}$: there are three cases to consider:
  - P_1 is fired alone: $P_1 \xrightarrow[B]{\alpha}_{\iota'} P_1'$, we have $Q = P_1' \,|\, (Q \setminus A)$, and $Q' = (P_1' \,|\, Q_1) \setminus A$, and $\forall C \in \mathcal{C}, \iota(C) = \iota'(C) \cup \mathsf{iA}^*_C(Q \setminus A)$. We can check that $\forall (C, L) \in B$, $\mathsf{iA}^*_C(Q_1) \cap \overline{L} = \mathsf{iA}^*_C(Q_1 \setminus A) \cap \overline{L}$, so the side-conditions of the two transitions $P \xrightarrow[B]{\alpha}_{\iota} Q$ and $P' \xrightarrow[B]{\alpha}_{\iota} Q'$ are equivalent;
  - Q_1 is fired alone: $Q_1 \xrightarrow[B]{\alpha}_{\iota'} Q_1'$, we have $Q = P_1 \,|\, (Q_1' \setminus A)$, and $Q' = (P_1 \,|\, Q_1) \setminus A$. We can check that the side-condition of the two transitions $P \xrightarrow[B]{\alpha}_{\iota} Q$ and $P' \xrightarrow[B]{\alpha}_{\iota} Q'$ are equivalent;
  - P_1 and Q_1 are fired together: $P_1 \xrightarrow[B_P]{\ell}_{\iota_P} P_1'$ and $Q_1 \xrightarrow[B_Q]{\overline{\ell}}_{\iota_Q} Q_1'$. We have $\alpha = \ell \,|\, \overline{\ell}$, and $B = B_P \cup (B_Q \setminus A)$, and $\forall C \in \mathcal{C}$, $\iota(C) = \iota_P(C) \cup \iota_Q(C) - (A \cup \overline{A})$, and $Q = P_1' \,|\, (Q_1' \setminus A)$, and $Q' = (P_1' \,|\, Q_1') \setminus A$. We can check that $P \xrightarrow[B]{\alpha}_{\iota} Q$ iff $P' \xrightarrow[B]{\alpha}_{\iota} Q'$ by checking that the side-conditions are equivalent;
- case $P = 0_{C_1} / C_2 \equiv 0_{C_1 - C_2}$: this process cannot progress;
- case $P = P' / C \equiv P'$ if $\mathcal{L}(P) \cap C = \{\}$: as $\mathcal{L}(P) \cap C = \{\}$, the blocking relation B and the prediction function ι remains the same when removing elements from C. Also, P' cannot perform a clock action in C, so we can conclude;
- case $P_1 / C_1 / C_2 \equiv P_1 / (C_1 \cup C_2)$: easy;

- case $P_1 \setminus A / C \equiv P_1 / C \setminus A$: easy;
- case $P \,|\, (Q / C) \equiv (P \,|\, Q) / C$ if $\mathcal{L}(P) \cap C = \{\}$: similarly as the $P_1 \,|\, (Q_1 \setminus A) \equiv (P_1 \,|\, Q_1) \setminus A$ case. The case where Q_1 is fired alone becomes more complicated: if $Q_1 \xrightarrow[B']{\alpha}_{\iota'} Q_1'$, proving that the side-conditions of the two possible derivations are equivalent becomes trickier. We have to check that $\forall (C', L) \in B'$, $\mathsf{iA}^*_{C'-C}(P) = \mathsf{iA}^*_{C'}(P)$, which is true because $\mathcal{L}(P) \cap C = \{\}$. Also, if α is a clock of C, the side-condition $\alpha \notin \mathsf{clocks}(P)$ is true for the same reason;
- case $\mathsf{p}(\vec{a}, \vec{\sigma}) \equiv P$ if $\mathsf{p}(\vec{a}, \vec{\sigma}) \stackrel{def}{=} P$: trivial. ◂

Now we show that processes able to perform a rendezvous action in the transition system can be rewritten into a canonical form.

Proof of Lemma 6 (Channel a).
By induction on the rules of the LTS.

- case (Act): Then $P = a{:}L.P'$, therefore $P \equiv ((a{:}L.P' + \mathsf{0}_{\mathsf{clocks}(P)}) \,|\, \mathsf{0}_{\{\}}) / \{\} \setminus \{\}$ and we can conclude;
- case (Sum_i): by induction hypothesis, $P \equiv ((a{:}L.Q + M) \,|\, \mathsf{0}_{\{\}}) / \{\} \setminus \{\} + N$, therefore $P \equiv ((a{:}L.Q + (M + N) \,|\, \mathsf{0}_{\{\}}) / \{\} \setminus \{\}$. Similarly, we can show that $P' \equiv (Q \,|\, \mathsf{0}_{\{\}}) / \{\} \setminus \{\}$, and the side-conditions are preserved;
- case (Com): impossible;
- case (Par_{lts}): by induction hypothesis, $P \equiv ((a{:}L.Q + M) \,|\, R) / C \setminus A \,|\, P_1$. We can assume, through alpha-conversion, that $\mathcal{L}(P_1) \cap C \cap (A \cup \overline{A}) = \{\}$, therefore $P \equiv ((a{:}L.Q + M) \,|\, (R \,|\, P_1)) / C \setminus A$ and $P' \equiv (Q \,|\, (R \,|\, P_1)) / C \setminus A$, and the side-conditions are preserved;
- case $(Restr_{\mathsf{lts}})$: by induction hypothesis, $P \equiv ((a{:}L.Q + M) \,|\, R) / C \setminus A_1 \setminus A_2$, therefore $P \equiv ((a{:}L.Q + M) \,|\, R) / C \setminus (A_1 \cup A_2)$, and $P' \equiv (Q \,|\, R) / C \setminus (A_1 \cup A_2)$, and the side-conditions are preserved;
- case $(Hide_{\mathsf{lts}})$: similarly;
- case (Con): by induction hypothesis. ◂

In a similar way, we can show that processes able to perform a clock action in the transition system can be rewritten into a canonical form.

Proof of Lemma 7 (Clock σ).
By induction on the rules of the LTS.

- case (Act): $P = \sigma{:}L.P'$, therefore $P \equiv ((\sigma{:}L.P' + \mathsf{0}_{\mathsf{clocks}(P)}) \,|\, \mathsf{0}_{\{\}}) / \{\} \setminus \{\}$ and we can conclude;
- case (Sum_i): by induction hypothesis, $P \equiv ((\sigma{:}L.Q + M) \,|\, \mathsf{0}_{\{\}}) / \{\} \setminus \{\} + N$, therefore $P \equiv ((\sigma{:}L.Q + (M + N) \,|\, \mathsf{0}_{\{\}}) / \{\} \setminus \{\}$. Similarly, we can show that $P' \equiv (Q \,|\, \mathsf{0}_{\{\}}) / \{\} \setminus \{\}$, and the side-conditions are preserved;
- case (Com): $P = P_1 \,|\, P_1'$, and by induction hypothesis, $P_1 \equiv ((\sigma{:}L_1.Q_1 + M_1) \,|\, \cdots \,|\, R_1) / C_1 \setminus A_1$, and $P_1' \equiv ((\sigma{:}L_1'.Q_1' + M_1') \,|\, \cdots \,|\, R_1') / C_1' \setminus A_1'$, and, through alpha-conversion, we can assume the hygiene condition, that is, C_1, C_1', A_1, A_1' are all disjoints, and free and bound variables are all disjoints. Therefore, $P \equiv ((\sigma{:}L_1.Q_1 + M_1) \,|\, \cdots \,|\, (\sigma{:}L_1'.Q_1' + M_1') \,|\, \ldots \,|\, (R_1 \,|\, R_1')) / (C_1 \cup C_1') \setminus (A_1 \cup A_1')$, and $P' \equiv (Q_1 \,|\, \cdots \,|\, Q_1' \,|\, \cdots \,|\, (R_1 \,|\, R_1')) / (C_1 \cup C_1') \setminus (A_1 \cup A_1')$. Using the hygiene condition, we can show the side-condition are preserved;

- case (Par_{lts}): by induction hypothesis,
  $P \equiv ((\sigma{:}L_1.Q_1 + M_1) \mid \cdots \mid R) / C \setminus A \mid R'$, and, through alpha-conversion, we can assume that $\mathcal{L}(R) \cap C \cap (A \cup \overline{(A)}) = \{\}$. Therefore
  $P \equiv ((\sigma{:}L_1.Q_1 + M_1) \mid \cdots \mid (R \mid R')) / C \setminus A$, and
  $P' \equiv (Q_1 \mid \cdots \mid (R \mid R')) / C \setminus A$. Using the hygiene condition, we can show the side-condition are preserved;
- case ($Restr_{\mathsf{lts}}$): easy;
- case ($Hide_{\mathsf{lts}}$): easy;
- case (Con): by induction hypothesis. ◂

Finally, we show that processes able to perform a τ action in the transition system can be rewritten into a canonical form, which look closely to what we have in the (Tau), (RdV) and ($Clock$) rules of the reduction system.

Proof of Lemma 8 (Silent action τ).
By induction on the rules of the LTS.

- case (Act): only the τ-action case apply here. Easy;
- case (Sum_i): only the τ-action case apply here. Easy;
- case (Com): only the rendezvous synchronisation case apply here. Using Lemma 6, we have that
  $P \equiv ((a{:}L_1.Q_1 + M_1) \mid R_1) / C_1 \setminus A_1 \mid ((\overline{a}{:}L_2.Q_2 + M_2) \mid R_2) / C_2 \setminus A_2$, therefore, assuming the hygiene condition,
  $P \equiv ((a{:}L_1.Q_1 + M_1) \mid (\overline{a}{:}L_2.Q_2 + M_2) \mid (R_1 \mid R_2)) / (C_1 \cup C_2) \setminus (A_1 \cup A_2)$, and
  $P' \equiv (Q_1 \mid Q_2 \mid (R_1 \mid R_2)) / (C_1 \cup C_2) \setminus (A_1 \cup A_2)$. Using the side-condition hypothesis of Lemma 6, we can check that the side-conditions hold;
- case (Par_{lts}): we have to distinguish the three cases. Easy;
- case ($Restr_{\mathsf{lts}}$): we have to distinguish the three cases. In the clock synchronisation case, if the premise of the rule does a clock action, we use Lemma 7 to conclude. The other cases are easy;
- case ($Hide_{\mathsf{lts}}$): we have to distinguish the three cases. Easy;
- case (Con): by induction hypothesis. ◂

Using Lemmas 5 and 8, we can show that, on the one hand, reductions correspond to unblockable τ-transitions, and that, on the other hand, unblockable τ-transitions correspond to reductions.

Proof of Lemma 9 (Harmony).

($\Rightarrow$) By induction on the TRS rules.

- Rule (Tau): easy;
- Rule (RdV): easy;
- Rule ($Clock$): we can show, by induction on n, that

  $$(\sigma{:}L_1.P_1 + M_1) \mid \cdots \mid (\sigma{:}L_n.P_n + M_n) \xrightarrow[\bigcup_{i \leqslant n} B_i]{\sigma} \iota_n \; P_1 \mid \cdots \mid P_n$$

  by defining $\forall C \in \mathcal{C}$, $\iota_i(C) = (\bigcup_{j \leqslant i} \mathsf{iA}(M_j)) - \{\sigma\}$ and $B_i = \{(\mathsf{clocks}(P_i), L_i)\}$. Then we can conclude;
- Rule (Par_{trs}): easy;
- Rule ($Restr_{\mathsf{trs}}$): easy;
- Rule ($Hide_{\mathsf{trs}}$): easy;
- Rule ($Congr$): using Lemma 5;

($\Leftarrow$) Using Lemma 8. Using the terms given in the statement of the Lemma, the three subcases are very similar:

- (τ action) We use the (*Congr*) rule, then the (*Tau*) rule. All the side-conditions are already given by Lemma 8, except $L \subseteq C \cup A \cup \overline{A}$. However, we are assuming that, $\forall (C, L) \in B$, $L = \{\}$. As $B = \{(\mathsf{clocks}(Q), L\} \,/\, C \setminus A$, we see that $L - C - (A \cup \overline{A}) = \{\}$. We can then conclude;
- (Rendezvous action) similarly;
- (Clock action) similarly. ◂