Strong Priority and Determinacy in Timed CCS

Luigi Liquori, Michael Mendler

Strong Priority and Determinacy in Timed CCS

Luigi Liquori
Inria, France

Michael Mendler
University of Bamberg, Germany

Abstract

Building on the classical theory of process algebra with priorities, we identify a new scheduling mechanism, called *(sequentially) constructive reduction* which is designed to capture the essence of synchronous programming. The distinctive property of this evaluation strategy is to achieve determinism-by-construction for multi-cast concurrent communication. In particular, it permits us to model shared memory multi-threading with reaction to absence as it lies at the core of the programming language Esterel. In the technical setting of CCS extended by clocks and priorities, we prove for a large class of processes, which we call *structurally coherent* the confluence property for constructive reductions. We further show that under some syntactic restrictions, called *pivotable* the operators of prefix, summation, parallel composition, restriction and hiding preserve structural coherence. This covers a strictly larger class of processes compared to those that are confluent in Milner's classical theory of CCS without priorities.

2012 ACM Subject Classification Theory of computation, Process algebras

Keywords and phrases Timed process algebras, determinacy, priorities, synchronous programming

Digital Object Identifier 10.4230/LIPIcs...

Funding *Luigi Liquori*: Partially funded by ETSI.
Michael Mendler: Partially funded by UniCA/I3S.

Acknowledgements The authors wish to thanks all the Synchron'23 participants for their useful comments. The second author is grateful to Martin Steffen and Gerald Lüttgen for helpful discussions.

1 Introduction

Classical process algebras, like CCS [22], CSP [16], Π-calculus [24, 25], just to mention a few, 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. This contrasts with automata theory where the interaction is single-pass: The environment provides a sequence of input actions to be fully consumed by the automaton before it outputs a result. This result is a decision value, determining if the string is accepted or not. The difference is reflected in the semantical theory:

- Process algebras are based on forms of bisimulation equivalence [20, 35], a game-theoretic notion. Two processes are equivalent if they define the same interaction game.
- Traditional automata theory is based on language equality, which is a functional model. Two programs are equivalent if they implement the same decision function.

These two modes of interaction are usually treated independently. Specifically, process algebras are typically applied for asynchronous systems, which are inherently non-deterministic. Functional models are used for synchronous systems which are inherently deterministic.

However, there are models of computation in which the game-theoretic and the functional mode are combined. In the domain of *Synchronous Programming*, processes interact with each other asynchronously at a micro-step level, yet synchronise in lock-step to advance jointly in iterative cycles of synchronous macro-steps. The micro-step interactions taking place during a macro-step determine the final outcome of the macro-step. This outcome defined at the point where all components pause to wait for a global logical clock. When

this clock arrives, all components proceed into the next computation cycle. Under the so-called *synchrony hypothesis*[1], the final outcome of a macro-step for each component is fully determined from the stimulus provided by the environment of that component during the micro-step interactions. As such, the interactions inside each subsystem at macro-step level (a game-theoretic process) can be abstracted into a global Mealy automaton (a deterministic IO-function). This synchronous model of computation has been very successful historically, replacing analogue and asynchronous circuits, dominating hardware design up to today. Synchronous models started with Statecharts [14] and has generated languages such as Signal [11], Lustre [13], Esterel [3], Quartz [31], SCCharts [36], just to mention a few.

In this paper we take a fresh look at the synchronous process algebras as a combination of game-theoretic and functional models. Specifically, we bring together two concepts for CCS that have previously been studied independently, viz. *clocks* and *priorities*. By strengthening the notion of *process priority* [7, 30, 27] in a natural way to what we call *strong priorities* or *precedence policies*, and combining this with the idea of a clock [12, 8, 26, 21], we obtain an adequate compositional setting for synchronous programming that reconciles non-determinism with functional reactions. This is surprising since many different hand-crafted semantics have been developed over the years but none of them fits into the classical framework of CCS.

1.1 Resolving Nondeterminism

Concurrency, by expression or by implementation, is both a convenient and unavoidable feature of modern software systems. Functional determinism is crucial for maintaining predictability and to manage design complexity by simple mathematical models [19, 6, 10]. If all primitive data manipulations are deterministic (as in the λ-calculus), then non-determinism arises either from unpredictable input data or the unpredictable timing of communication actions. External non-determinism that is resolved by the input data is innocuous. A process that waits for input and then branches control depending on the value of the input is functionally predictable. If the input can arrive from different concurrent senders, then non-determinism arises if the external timing is unobservable. For instance, the behaviour of a shared data object, in general, may depend on the order in which two method calls are executed on it. This creates non-determinism if the actions do not commute, i.e. if the object's state transitions are not *confluent*. The three standard ways to resolve data races and achieve confluence are *isolation*, *cloning* and *precedence*. Let us discuss these briefly. If actions α_1 and α_2 affect two concurrently independent parts of Q, say, $Q \stackrel{def}{=} \alpha_1.Q_1 \,|\, \alpha_2.Q_2$, then commutation is guaranteed. The parallel composition $P \,|\, Q \,|\, R$ ends up in configuration $P_1 \,|\, Q_1 \,|\, Q_2 \,|\, R_2$ no matter which of P or R communicates first, and with which of the two concurrent parts of $\alpha_1.Q_1$ or $\alpha_2.Q_2$ of Q they synchronise with if α_1 and α_2 are indistinguishable. This is the method of *isolation* which is captured naturally by the parallel operator of process algebras. Another solution is to *clone* a resource unboundedly often, via a repetition operator. If $\alpha_1{=}\alpha{=}\alpha_2$ and $Q \stackrel{def}{=} \alpha^*.Q'$ means the prefix α is repeated infinitely often, i.e., $Q \xrightarrow{\alpha} (Q' \,|\, \alpha^*.Q')$, then Q will offer α to both processes P and R independently by cloning the residual Q'. The parallel composition $P \,|\, Q \,|\, R$ will always end up in configuration $P_1 \,|\, Q' \,|\, Q' \,|\, Q \,|\, R_2$ when both P and R have finished their communication with Q. In all cases, the shared object Q does not really care about the ordering in which α_1 and α_2 are executed. Yet, what if the behaviour of a process Q must depend on the choice between α_1 and α_2? Then, the canonical solution is to enforce a *scheduling precedence* on the actions of the

[1] The "synchrony hypothesis" assume that the reaction of a component to input happens faster than the environment is able to produce new stimulus: the time needed to compute the reaction is unobservable by the environment. Also known as the "zero-time" assumption [17] or "fundamental mode operation" [33].

environment in order to preserve coherence of the resource. Technically, if $Q = \alpha_1.Q_1 + \alpha_2.Q_2$ represents a conflicting choice of actions α_i leading to distinct states Q_i, then we may restrict the concurrent environment $E = P \,|\, [\cdot] \,|\, R$ in the following ways:

- If Q is a store where $\alpha_1 = a?v$ is a data input and $\alpha_2 = a!w$ an output, we want the input take *precedence* over the output to implement data-flow causality. Let us write $Q \xrightarrow{\alpha_2}_{\{\alpha_1\}} Q_2$ to express that the output α_2 only synchronises with a matching $\overline{\alpha_2}$ action if the environment E does not offer to engage in an input $\overline{\alpha_1}$ at the same time. In contrast, the input action $Q \xrightarrow{\alpha_1}_{\{\}} Q_1$ is not blocked by the output α_2.
- Sometimes an ordering does not make sense, say if $\alpha_1 = a?v$ and $\alpha_2 = a?w$ are inputs of distinct values $v \neq w$ to be received by Q. Since these actions make Q take on distinct continuation states $Q_1 \neq Q_2$, the environment E must be blocked as it attempts to engage in both α_1 and α_2, concurrently from P and from R. This can be modelled as a precedence of α_1 over α_2 and α_2 over α_1 which generates transitions $Q_2 \xleftarrow{\alpha_2}_{\{\alpha_1\}} Q \xrightarrow{\alpha_1}_{\{\alpha_2\}} Q_1$.

As such, precedences can be used to resolve the timing uncertainty about which one of two distinct (conflicting) action steps is taken by a shared process and further, for this action step, resolve the uncertainty about which of several concurrent processes in its environment is consuming it. In this paper we use CCS as our formalism to study the use of precedences for achieving determinism. Similar ideas might be possible for other process algebras like SCCS, CSP or the π-calculus. We find CCS most plausible as a point of departure since the concept of priorities (which is very related) is already well established in CCS. Also, in CCS, the combination of synchronous communication with consumable prefixes and asynchronous scheduling via local rendezvous synchronisation makes the problem of generating determinacy more prominent. Isolation and cloning are well-known techniques to achieve determinism in process algebras, see e.g., Chap. 11 of [22]. However, scheduling priorities do not seem to have been systematically used for that purpose. In this paper we propose *precedence policies* as a refinement of existing priority schemes for CCS for achieving determinism. Similar ideas might be possible for other process algebras like SCCS, CSP or the π-calculus.

Few papers extending CCS have been of great inspiration of this paper, namely the Camilleri-Winskell $\mathsf{CCS}^{\mathsf{CW}}$ [7], the Phillips $\mathsf{CCS}^{\mathsf{Ph}}$ [27], and and Cleaveland-Lüttgen-Natarajan $\mathsf{CCS}^{\mathsf{CLN}}$ [30] extensions of CCS with some priority choices, together with Chap. 11 of Milner's book introducing the CCS dialect $\mathsf{CCS}^{\mathsf{cc}}$. A fairly informal presentation and comparison of those algebras can be found in App. A. Nevertheless, none of them has been able to reach full confluence, *conditio sine qua non* to set up a solid basis for semantic of synchronous programming. This paper argues that adding broadcasting actions like *synchronous clocks* to the above proposals is a sufficient condition to reach confluence. *Clocks* are a well-known notion from Timed Process Algebras, like e.g. TPL [15], PMC [12] and CSA [8]. Intuitively, a clock σ is a "broadcast action" in the spirit of Hoare's CSP [16] that acts as a synchronisation barrier for a set of concurrent processes. It bundles each process' actions into a sequence of *macro-steps* that are aligned with each other in a lock-step fashion. During each macro-step, a process executes only a temporal slice of its total behaviour, at the end of which it waits for all other processes in the same *clock scope* to reach the end of the current phase. When all processes have reached the barrier, they are released into the next round of computation. This scheduling principle is well-known from sequential circuits [33] and VHDL [17], or synchronous programming like Esterel [3], Lustre [13], SCCharts [36], BSP [34], just to mention a few. For our purposes, clocks are the universal trick to break causality cycles in priority scheduling, when processes need to communicate with each other in iterated cycles (so-called *ticks* or *macro-steps*) while maintaining determinacy. The idea is that the clock should fire only if the system has stabilised and no other admissible data actions are possible

(*maximal progress*) is a standard assumption in timed process algebras [15, 8]. Combining prioritised rendez-vous with broadcast actions in a Plotkin SOS style and discovering that an interleaving of micro- and macro-steps leads to confluence is the main result of this paper.

In the next sections, we present the syntax, the structural operational semantics à la Plotkin, we set up all the definitions and the metatheory necessary to formally prove the claimed informal properties. For lack of space, three separate appendices will present how research towards determinism for process algebras pass through those above cite papers, a collection of more tricky examples, and the full proofs.

2 A Process Algebra with Clocks and Priorities

We call our clocked process algebra "SynPaTick" (denoted by $\mathsf{CCS}^{\mathsf{spt}}$), whose syntax and operational semantics are defined in the subsections below. In a nutshell, $\mathsf{CCS}^{\mathsf{spt}}$ is an extension of Milner's CCS process algebra with priorities à la Phillips' $\mathsf{CCS}^{\mathsf{Ph}}$ and broadcasted clock actions à la Hoare's CSP and as in timed algebras TPL [15] and CSA [8].

2.1 $\mathsf{CCS}^{\mathsf{spt}}$ Syntax

The terms in $\mathsf{CCS}^{\mathsf{spt}}$ form a set $\mathcal{P}$ of *agents*. We use $P, Q, R, S \ldots$ to range over $\mathcal{P}$. Let $\mathcal{A}$ be a countably infinite set of *channel names* with $a, b, c \ldots$ ranging over $\mathcal{A}$. As in CCS, the set $\overline{\mathcal{A}}$ is a disjoint set of *co-names* with elements denoted by $\overline{a}, \overline{b}, \overline{c} \ldots$ that are in bijection with $\mathcal{A}$; in other words, the overbar operator $\bar{\cdot}$ switches between $\mathcal{A}$ and $\overline{\mathcal{A}}$, i.e., $\overline{\overline{a}} = a$ for all $\overline{a} \in \mathcal{A}$. We will refer to the names $a \in \mathcal{A}$ as *input* (or *receiver*) labels while the co-names $\overline{a} \in \overline{\mathcal{A}}$ denote *output* (or *sender*) labels. Let $\mathcal{C}$ be a countably infinite set of broadcasted *clock names*, disjoint from both $\mathcal{A}$ and $\overline{\mathcal{A}}$ and let σ range over $\mathcal{C}$. Every clock acts as its own co-name, $\overline{\sigma} = \sigma$ and so each $\sigma \in \mathcal{C}$ is also considered as a *clock label*. Let $\mathcal{L}$ be the set of input, output and clock *labels* defined as $\mathcal{A} \cup \overline{\mathcal{A}} \cup \mathcal{C}$ and let ℓ range over $\mathcal{L}$, and let L, H range over subsets of $\mathcal{L}$. We write $\overline{L}$ for the set $\{\overline{\ell} \mid \ell \in L\}$. Let Act be the set of all *actions* defined as $\mathcal{L} \cup \{\tau\}$ by adjoining the *silent* action $\tau \notin \mathcal{L}$ and let α range over Act. Viewed as actions, the input and output labels $\ell \in \mathcal{R} \stackrel{def}{=} \mathcal{A} \cup \overline{\mathcal{A}} \subseteq Act$ are referred to as *rendez-vous* or *channel actions* to distinguish them from the *clock actions* $\sigma \in \mathcal{C} \subseteq Act$ which also called *broadcast actions*. Further, to distinguish the elements of $\mathcal{R} \cup \mathcal{C}$ from τ as a subset of Act we call the former *visible* actions in contrast to τ that is the *invisible* action. All symbols can appear indexed. The process terms $\mathcal{P}$ are defined by the following abstract syntax:

P, Q, R, S	::=	$\mathsf{0}$	stop (inaction)
	|	$\alpha{:}H.P$	action prefix with $\alpha \in \mathcal{L}$ and $H \subseteq \mathcal{L}$
	|	$P + Q$	choice
	|	$P \mid Q$	parallel composition
	|	$P \setminus L$	restriction
	|	P / L	hiding
	|	A	process name $A \in \mathcal{I}$

Intuitively, $\mathsf{0}$ denotes the *inactive* process which stops all computations, including broadcasting synchronisation actions. The *action prefix* $\alpha{:}H.P$ denotes a process that offers to engage in the action α and then behave as P while the *blocking* set $H \subseteq \mathcal{L}$ lists all actions taking precedence over α. A prefix with $\alpha \in \mathcal{R}$ denotes a CCS-style rendez-vous action. In case $\alpha \in \mathcal{C}$, it denotes a CSP-style broadcast action that can communicate with all the surrounding processes sharing on the same clock only when the clock is universally *consumed*. The process $A \in \mathcal{I}$, where $\mathcal{I}$ is a set of *process identifiers*, refers to a process whose behaviour is specified indirectly by a definitional equation $A \stackrel{def}{=} P$. We assume that in every restriction expression $P \setminus L$ the set $L \subseteq \mathcal{R}$ consists of rendez-vous actions, and in every hiding expression P / L we have $L \subseteq \mathcal{C}$. These *scoping* operators act as name binders that introduce local scopes.

The acute reader surely notices the absence of the relabelling operator $P[f]$, useful in Milner's CCS to define the well-known "standard concurrent form" $(P_1[f_1] \mid \cdots \mid P_2[f_n]) \setminus L$. Because all of our examples are captured without need of relabelling, and for the sake of simplicity, relabelling will be treated in the full version of this work.

A useful abbreviation is to drop the blocking sets if they are empty or drop the continuation process if it is inactive. For instance, we will typically write $a.P$ instead of $a{:}\{\}.P$ and $a{:}H$ rather than $a{:}H.\mathsf{0}$. Also, it is useful to drop brackets for the blocking set if it is a singleton, writing $a{:}b.P$ instead of $a{:}\{b\}.P$. With these notational shortcuts we can naturally take each action $\alpha \in \mathcal{L}$ also as a prefix process $\alpha{:}\{\}.\mathsf{0}$. In this way, each action α is a process that offers to engage in α, without blocking, and then stops.

In our concrete notation for expressions using the abstract syntax of processes, we will assume that the binary operator $+$ is right associative and taking higher binding power than the binary operator $\mid$. Thus, we write $P + Q + R \mid S$ instead of $(P + (Q + R)) \mid S$. Also, we assume that the unary operators of prefix, restriction and hiding have higher binding power than the binary operators. Hence, we write $P \setminus c + a.b.c.Q \mid R / \sigma$ instead of $((P \setminus c) + (a.(b.(c.Q)))) \mid (R / \sigma)$. The *sub-processes* of a P are all sub-expressions of P and (recursively) all sub-processes of Q for definitions $A \stackrel{def}{=} Q$ where A is a sub-expression of P. Sometimes it is useful to consider processes in a fragment of $\mathsf{CCS}^{\mathsf{spt}}$. Specifically, a process is called *unclocked* if P does not contain any clock prefix $\sigma{:}H.Q$ as a sub-process. A process is called *sequential* or *single-threaded* if it does not contain any parallel sub-process $Q \mid R$. P is *action-closed*, or simply *closed*, if if all sub-processes $\ell{:}H.Q$ in P occur in the scope of a restriction or hiding operator that binds ℓ. We write $\mathsf{vA}(P)$ for the set of free ("visible") action names of P. We identify syntactically two processes that differ only in the choice of bound names, and we work under the *Barendregt's hygiene convention* [1].

Following Milner's CCS description in Chapter 3 of [22] and Milner's presentation of the π-calculus in [23], we define a structural congruence over processes. The idea of the structural congruence is to split interaction rules between agents from rules governing the "left-to-right" representation of expressions. The idea of introducing an equivalence of processes via a set of equations (congruence) is not new and it has many advantages: it allows to use those equations in the derivation rules of the SOS making the latter simpler and easy to use, and it allows to set up formal equivalence of processes increasing for each program the number of legal SOS reductions, that, otherwise would be blocked (non reducible). Milner divides CCS equations in static, dynamic and expressions laws: we will choose a modern approach inspired by the pioneering work of Berry and Boudol [5].

▸ **Definition 1** (Structural Congruence $\equiv$)**.**
Structural congruence, written $\equiv$, is the smallest congruence over $\mathcal{P}$ that satisfies the equations in Fig. 1, where $\star \in \{\mid, +\}$ and $\wr \in \{\setminus, /\}$:

Note that we have dropped from our structural rules all of Milner's τ-laws, i.e. the ones related to the presence of invisible actions, like e.g. $\alpha.\tau.P \equiv \alpha.P$, or $P + \tau.P \equiv \tau.P$, and we have also dropped the congruence equations related to the uniqueness of solution of recursive equations and the *expansion laws* related to the *standard concurrent form* (cf. Chap. 3, Prop. 2, Prop. 4(2) and 4(5) of [22], respectively). This is not surprising. It is known that timed process algebras and process algebras with priorities enjoy extra expressiveness that invalidates some of the familiar dynamic laws, see e.g., [12, 15, 8, 30, 27]. $\mathsf{CCS}^{\mathsf{spt}}$ shares features (clocks and priorities) with timed and prioritised extensions of CCS. Hence, we must take a conservative approach and only consider a minumum set of purely structural rules that do not impinge on the dynamics of expression behaviour.

$$
\begin{array}{lrcll}
(\mathsf{Const}) & A & \equiv & P & \text{if } A \stackrel{def}{=} P \\
(\mathsf{Comm}) & P \star Q & \equiv & Q \star P & \\
(\mathsf{Assoc}) & (P \star Q) \star R & \equiv & P \star (Q \star R) & \\
(\mathsf{Zero}) & P \star 0 & \equiv & P & \\
(\mathsf{Idem}_{+}) & P + P & \equiv & P & \\
(\mathsf{Scope}_{\wr}) & P \wr L \wr L' & \equiv & P \wr (L \cup L') & \\
(\mathsf{Scope}_{0}) & 0 \wr L & \equiv & 0 & \\
(\mathsf{Scope}_{\alpha}) & (\alpha{:}H.P) \setminus L & \equiv & \begin{cases} 0 & \text{if } \alpha \in L \cup \overline{L} \\ \alpha{:}H.(P \setminus L) & \text{otherwise} \end{cases} & \\
(\mathsf{Scope}_{\setminus}) & (P \star Q) \setminus L & \equiv & \begin{cases} P \star Q \setminus L & \text{if } \mathsf{vA}(P) \cap (L \cup \overline{L}) = \{\} \\ P \setminus L + Q \setminus L & \text{if } \star = + \\ P \setminus L \mid Q \setminus L & \text{if } \star = \mid \text{ and } \mathsf{vA}(P) \cap \overline{\mathsf{vA}(Q)} \cap (L \cup \overline{L}) = \{\} \end{cases} &
\end{array}
$$

Figure 1 Structural Congruence.

$$
\frac{}{\alpha{:}H.P \xrightarrow[0]{\alpha}_H P}\ (Act) \qquad \frac{P' \xrightarrow[R']{\alpha}_H Q' \quad P \equiv P' \quad Q \equiv Q' \quad R \equiv R'}{P \xrightarrow[R]{\alpha}_H Q}\ (Struct)
$$

$$
\frac{P \xrightarrow[R]{\alpha}_H Q \quad L' = L \cup \overline{L} \quad \alpha \notin L' \quad H' = H - L'}{P \setminus L \xrightarrow[R \setminus L]{\alpha}_{H'} Q \setminus L}\ (Restr) \qquad \frac{P \xrightarrow[R]{\alpha}_H Q \quad H' = H - L}{P \,/\, L \xrightarrow[R/L]{\alpha/L}_{H'} Q \,/\, L}\ (Hide)
$$

$$
\frac{P \xrightarrow[R]{\alpha}_H P' \quad \alpha \notin \mathcal{C}}{P \mid Q \xrightarrow[R \mid Q]{\alpha}_H P' \mid Q}\ (Par_{1,2}) \qquad \frac{P \xrightarrow[R]{\alpha}_H P'}{P + Q \xrightarrow[R]{\alpha}_H P'}\ (Sum_{1,2})
$$

$$
\frac{P \xrightarrow[R_1]{\ell}_{H_1} P' \quad Q \xrightarrow[R_2]{\overline{\ell}}_{H_2} Q' \quad H = race(P \mid Q)}{P \mid Q \xrightarrow[R_1 \mid R_2]{\ell \mid \overline{\ell}}_{H_1 \cup H_2 \cup H} P' \mid Q'}\ (Par_3)
$$

where

$$
\alpha \,/\, L \stackrel{def}{=} \begin{cases} \tau & \text{if } \alpha \in L \\ \alpha & \text{otherwise} \end{cases} \qquad \ell \mid \overline{\ell} \stackrel{def}{=} \begin{cases} \tau & \text{if } \ell \in \mathcal{R} \\ \sigma & \text{if } \ell = \sigma \end{cases}
$$

$$
race(P \mid Q) \stackrel{def}{=} \begin{cases} \{\tau\} & \text{if } H_1 \cap \overline{\mathsf{iA}}(Q) \nsubseteq \{\ell\} \text{ or } H_2 \cap \overline{\mathsf{iA}}(P) \nsubseteq \{\overline{\ell}\} \\ \{\} & \text{otherwise} \end{cases}
$$

Figure 2 The $\mathsf{CCS}^{\mathsf{spt}}$ Plotkin's Structural Operational Semantics.

2.2 Operational Semantics

The operational semantics for $\mathsf{CCS}^{\mathsf{spt}}$ is given using a labelled transition relation $P \xrightarrow{\alpha} Q$ in the style of Plotkin's Structured Operational Semantics (SOS) [29, 28], where $\alpha \in Act$ is the action that labels the SOS transition. When $\alpha \in \mathcal{L}$ the transition corresponds to a communication step, namely a rendez-vous action or a broadcasted clock action. When $\alpha = \tau$ is silent, then the transition corresponds to a *reduction* or an *evaluation* step. In addition, we will decorate the nondeterministic SOS with annotations that provides sufficient information to express a uniform and confluent reduction strategy based on scheduling precedences. We will call this new reduction strategy *(sequentially) constructive scheduling*. Including these annotations, our SOS judgment takes the form $P \xrightarrow[R]{\alpha}_H Q$ where $P, Q, R \in \mathcal{P}$ are processes, $\alpha \in Act$ is an action, and $H \subseteq Act$ a set of actions. We call the three composite labels α, H, and R a *c-action*, written $\alpha{:}H[R]$. The information provided by such a c-action is sufficient to define our *constructive reduction* strategy for $\mathsf{CCS}^{\mathsf{spt}}$ that satisfies a Church-Rosser property.

$$
\begin{array}{rcl}
\mathsf{iA}(0) & \stackrel{def}{=} & \{\,\} \\
\mathsf{iA}(\alpha{:}H.P) & \stackrel{def}{=} & \{\alpha\} \\
\mathsf{iA}(P+Q) & \stackrel{def}{=} & \mathsf{iA}(P) \cup \mathsf{iA}(Q) \\
\mathsf{iA}(P \,|\, Q) & \stackrel{def}{=} & (\mathsf{iA}(P) \cup \mathsf{iA}(Q)) - \mathcal{C} \cup \{(\ell \,|\, \overline{\ell}) \mid \ell \in \mathsf{iA}(P) \cap \overline{\mathsf{iA}(Q)}\} \\
\mathsf{iA}(P \setminus L) & \stackrel{def}{=} & \mathsf{iA}(P) - (L \cup \overline{L}) \\
\mathsf{iA}(P \,/\, L) & \stackrel{def}{=} & \mathsf{iA}(P) - L \cup \{\tau \mid L \cap \mathsf{iA}(P) \neq \{\,\}\} \\
\mathsf{iA}(A) & \stackrel{def}{=} & \mathsf{iA}(P) \qquad \text{where } A \stackrel{def}{=} P
\end{array}
$$

Figure 3 Strong Initial Actions.

We will use $P \xrightarrow{\alpha} Q$ as an abbreviation to state that there is a transition for some R and H.

The semantics is defined inductively as the smallest relation closed under rules in Fig. 2. We call each transition derivable by Fig. 2 an *admissible* transition. The intended meaning of our annotated semantics for admissible transitions is as follows: α is the usual action "passed back" in the SOS, H (also called *blocking set*) is the set of actions that block the transition, and R (also called *concurrent context*) is a process that represents the behaviour of all threads in P that execute concurrently with the transition: note that actions in R are potentially in competition with α. The set H represents the resources that are consumed by the action and of R as the concurrent context that potentially competes for H. In a parallel composition $P \,|\, Q$ of P with another process Q, the context R of might interact with the environment Q to generate actions in the blocking set H. If this happens, the reduction in P with blocking set H will be considered blocked in our scheduled semantics (see Def. 4).

In rules of Fig. 2 the reader can observe how the annotation for the concurrent context and the blocking sets are inductively built. In a nutshell: rule (*Act*) is the axiom annotating in the transition the action and the (empty) environment. Rules (*Restr*) and (*Hide*) deal with "local restriction" of rendez-vous actions and "local hiding" of broadcasted clocks. Rules ($Par_{1,2}$) and ($Sum_{1,2}$) are almost standard and deal with parallelism and sum. Rule (*Struct*) internalises in the transition the above presented structural equivalence of processes. Rule (Par_3) is compositional as one can expect for both rendez-vous and broadcast, and it have the extra *race* condition enriching the blocking set of the conclusion in case P can interfere with Q and vice-versa. Note that if we ignore the scheduling labels, then the operational semantics of unclocked processes coincides precisely with the operational semantics of CCS. The rules for clock actions coincide with the semantics of CSP broadcast actions.

2.3 Constructive Scheduling

The scheduling mechanism of $\mathsf{CCS}^{\mathsf{spt}}$ provides a new form of *constructive evaluation* that satisfies a confluence property for a large class of processes, the so-called *coherent* processes. To connect our approach with that of $\mathsf{CCS}^{\mathsf{Ph}}$ and $\mathsf{CCS}^{\mathsf{CW}}$ we first consider the following definition of a weaker form of scheduling based on the strong initial actions of a process.

▸ **Definition 2** (Strong Initial Actions). *The set $\mathsf{iA}(P) \subseteq Act$ of* strong initial *actions of a process P is the smallest set closed under the inductive rules in Fig. 3.*

The traditional approach in prioritised process algebras [7, 30, 27] is to block the transitions of a process P by its strong initial actions $\mathsf{iA}(P)$ as captured by the following Def. 3.

▸ **Definition 3** (Weakly Enabled Transitions). *An admissible transition $P \xrightarrow[R]{\alpha}_H Q$ is called* weakly enabled *if $H \cap (\overline{\mathsf{iA}}(R) \cup \{\tau\}) = \{\,\}$.*

Let us call a process P *free* if the blocking sets of all action prefixes $\alpha{:}H.Q$ occurring as sub-processes of P have an empty blocking set, i.e. $H = \{\,\}$. The presence of blocking information

makes weakly enabled transitions become a proper subset of admissible transitions. For general non-free processes, weak enabling prunes certain transitions and thus implements a priority-based scheduling regime. In order to implement our stronger precedence-based scheduling we need to look at the *weak initial actions* for a process.

▸ **Definition 4** (Weak Initial Actions & Strong Enabling). *The set* $i\mathsf{A}^\tau(P)$ *of* weak initial *transitions is the smallest extension* $i\mathsf{A}(P) \subseteq i\mathsf{A}^\tau(P)$ *such that if* $\alpha \in i\mathsf{A}^\tau(Q)$ *and* $P \xrightarrow{\tau} Q$ *then* $\alpha \in i\mathsf{A}^\tau(P)$. *Let* $\overline{i\mathsf{A}}^\tau(P) = \overline{i\mathsf{A}^\tau(P)}$. *A transition* $P \xrightarrow[R]{\alpha}_H P'$ *is called* strongly enabled *if* $H \cap (\overline{i\mathsf{A}}^\tau(R) \cup \{\tau\}) = \{\}$.

By looking at the definition, it is easy to see that $\mathsf{iA}(P) \subseteq \mathsf{iA}^\tau(P)$, so every strongly enabled transition is weakly enabled: as such, for free processes, the notions of weak and strong enabling coincide. Note also that enabling as a scheduling constraint is not monotonic under parallel composition. The enabledness property of a transition is not, in general, preserved when the process is placed into a concurrent context. If a reduction $P \xrightarrow[R]{\tau}_H P'$ is enabled, then we know that $H \cap \overline{\mathsf{iA}}(R) = \{\}$ or $H \cap \overline{\mathsf{iA}}^\tau(R) = \{\}$, depending on what scheduling we choose. This means that the context R cannot (weakly) generate a synchronisation with a blocking initial action. This does not mean that an extension $R \,|\, Q$ may not perhaps have $H \cap \overline{\mathsf{iA}}(R \,|\, Q) \neq \{\}$ or $H \cap \overline{\mathsf{iA}}^\tau(R \,|\, Q) \neq \{\}$. This happens if blocking actions come from Q or, in the case of strong enabling, become reachable though the interactions of R and Q. Then the parallel extension $P \,|\, Q$ does not permit a reduction $P \,|\, Q \xrightarrow[R]{\tau}_H P' \,|\, Q$.

We are interested in exploiting priority-based scheduling to achieve determinacy of reduction. According to Milner [22], the notion of determinacy is tied up with predictability: *"if we perform the same experiment twice on a determinate system, starting each time in its initial state, then we expect to get the same result, or behaviour, each time."* The strongest form of determinacy is the notion of structural determinism.

▸ **Definition 5.** *A process* P *is* structurally deterministic *for an action* $\alpha \in Act$, *if for every derivative* Q *of* P, *if* $Q \xrightarrow{\alpha} Q_1$ *and* $Q \xrightarrow{\alpha} Q_2$ *we have* $Q_1 \equiv Q_2$.

Structural determinism disallows processes such as $a.A + a.B$ for $A \not\equiv B$ or parallel compositions such as $a.A \,|\, a.B$ where an action a is used in more than one prefix of a single thread or of concurrent threads. The former violates predictability as it introduces externally uncontrollable non-determinism. We don't exclude the latter, as it would prevent multi-receiver and multi-sender applications. For silent actions, structural determinism is too strong. We are not interested in the uniqueness of the immediate τ-redexes of a process but the uniqueness of the final normal form. As such, the notion of *confluence* is more useful.

▸ **Definition 6** (Structural Confluence). *A process* P *is* structurally confluent *if for every derivative* Q *of* P *and reductions* $Q \xrightarrow{\tau} Q_1$ *and* $Q \xrightarrow{\tau} Q_2$ *with* $Q_1 \not\equiv Q_2$, *there exist* Q' *such that* $Q_1 \xrightarrow{\tau} Q'$ *and* $Q_2 \xrightarrow{\tau} Q'$.

A process P is in *normal form* if has no admissible τ-transition. Then all normal forms of a structurally confluent process must be structurally equivalent. The main result of the paper (Thm. 17 together with Propositions 47–57) is to identify a large class of processes, called *policy-coherent* processes, which are structurally confluent under *c-enabled* reductions. The notion of policy-coherence is a refinement of Milner's notion of confluence ([22] Chap. 11). The reduction strategy of c-enabledness implements a form of *constructive scheduling* that refines strong enabling (Def. 4) discussed above. It turns out that each policy-coherent process, under constructive scheduling, is *clock-deterministic* (Prop. 19) and satisfies *maximal*

progress (Prop. 24). No matter in which order we execute c-enabled reductions, when the normal form is reached, and only then, a clock can tick. Since the normal form is uniquely determined by confluence, the next state reached by each clock tick is uniquely determined. Thus, each policy-coherent process represents a "synchronous stream". As such, our work extends the classical non-deterministic theory of timed (multi-clocked) process algebras [12, 15, 8] for applications in deterministic synchronous programming. Before we expound on the scheduling theory of $\mathsf{CCS}^{\mathsf{spt}}$ in Sec. 4, let us discuss some examples.

3 Examples

We begin with some simple examples to illustrate the operation of our scheduling semantics. The focus is on how the transitions labels, blocking sets and action environment, implement precedences.

▸ **Example 7** (Read Before Write)**.** Let us see how the rendez-vous synchronisation is scheduled by the precedences that come with a prioritised choice. Consider a "store" process $S \stackrel{def}{=} w{+}r{:}w$ that offers a write action w and a read action r but prefers the write over the read. If we put S in parallel with a reader $R \stackrel{def}{=} \overline{r}$ we can let read actions r and $\overline{r}$ synchronise in a reduction $S \,|\, R \xrightarrow[0]{\tau}_{\{w\}} 0$. The fact that w takes priority over r is recorded in the blocking set $\{w\}$ which contains the action w. This will block R's transition when a parallel writer $W \stackrel{def}{=} \overline{w}$ is added. Specifically, the reduction $w + r{:}w \,|\, \overline{r} \,|\, \overline{w} \xrightarrow[\overline{w}]{\tau}_{\{w\}} \overline{w}$ is not weakly enabled, because $\{w\} \cap \overline{\mathsf{iA}}(\overline{w}) = \{w\} \cap \{w\} \neq \{\,\}$. The only enabled reduction of process $S \,|\, R \,|\, W$ is to let S and W synchronise to produce the transition $w + r{:}w \,|\, \overline{r} \,|\, \overline{w} \xrightarrow[\overline{r}]{\tau}_{\{w\}} \overline{r}$ which is even strongly enabled, since $\{w\} \cap \overline{\mathsf{iA}}^*(\overline{r}) = \{\,\}$.

▸ **Example 8** (Two Thread Process)**.** The two-thread process $a{:}b{+}\overline{b} \,|\, \overline{a}{+}b{:}a$ has two admissible transitions under (Par_3) for $\tau.0$ through $a{:}b \,|\, \overline{a}$ or $\overline{b} \,|\, b{:}a$. However, since $b \in \overline{\mathsf{iA}}(a{:}b + \overline{b})$ and $a \in \overline{\mathsf{iA}}(\overline{a} + b{:}a)$ both the actions $a{:}b$ and $b{:}a$ are blocked.

A more subtle form of blocking occurs for strong enabling. Two parallel processes can create a circular dependency through the priorities of their initial actions, e.g., when accessing a third process that is acting as a resource. The following example simulates two processes that wait for each other to write to different memory cells, before they can write to the memory cell on which the other process is waiting.

▸ **Example 9** (Deadlock)**.** Consider the process $S \stackrel{def}{=} w_0 + r_0{:}w_0 \,|\, w_1 + r_1{:}w_1$ which offers receiver actions w_i and r_i for $i \in \{0,1\}$ such that w_i has precedence over r_i. Now take processes $P_0 \stackrel{def}{=} \overline{r_0}.\overline{w_1}$ and $P_1 \stackrel{def}{=} \overline{r_1}.\overline{w_0}$ which first aim to synchronise with r_i and then sequentially afterwards with w_{1-i}. The parallel composition $P_0 \,|\, S \,|\, P_1$ has two initial τ-reductions $P_0 \,|\, w_0 + r_0{:}w_0 \,|\, \overline{w_0} \xleftarrow[P_0 \,|\, (w_0 + r_0{:}w_0)]{\tau}_{\{w_1\}} P_0 \,|\, S \,|\, P_1 \xrightarrow[w_1 + r_1{:}w_1 \,|\, P_1]{\tau}_{\{w_0\}} \overline{w_1} \,|\, w_1 + r_1{:}w_1 \,|\, P_1$. These reductions are both weakly enabled. If we continue under weak enabling then the residual process $\overline{w_1} \,|\, w_1 + r_1{:}w_1 \,|\, P_1$ for the first reduction must first execute the synchronisation on w_1 and thus proceed as $\overline{w_1} \,|\, w_1 + r_1{:}w_1 \,|\, P_1 \xrightarrow{\tau} P_1$ while the residual process $P_0 \,|\, w_0 + r_0{:}w_0 \,|\, \overline{w_0}$ can only reduce to P_0. Hence, the resulting final outcome of executing $P_0 \,|\, S \,|\, P_1$ is non-deterministic. Observe that none of the two reductions displayed above is actually strongly enabled. For the right reduction we find that its concurrent environment $R_0 \stackrel{def}{=} w_1 + r_1{:}w_1 \,|\, P_1$ has a reduction sequence $R_0 \xrightarrow{\tau} \overline{w_0}$ and thus $\{w_0\} \cap \overline{\mathsf{iA}}^*(R_0) \subseteq \{w_0\} \cap \overline{\mathsf{iA}}^*(\overline{w_0}) = \{w_0\} \cap \{w_0\} \neq \{\,\}$. Symmetrically, for the concurrent environment $R_1 \stackrel{def}{=} P_0 \,|\, w_0 + r_0{:}w_0$ of the left reduction we have $R_1 \xrightarrow{\tau} \overline{w_1}$ and hence $\{w_1\} \cap \overline{\mathsf{iA}}^*(R_1) \neq \{\,\}$.

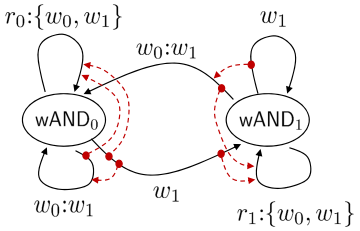
The deadlocking processes of Ex. 9 have the special property that priorities only exist between receiver actions, while sender actions never appear in any choice contexts. These are the

processes covered by $\mathsf{CCS}^{\mathsf{CW}}$ [7] which restricts prioritised choice to receiver actions in the same way as the programming language Occam [2] with its "PRIALT" construct does.

▸ **Example 10** (Read/Write Memory). Let w_0, w_1, r_0, r_1 the labels of writing/reading a boolean value 0 or 1, respectively. A memory would be modelled by two mutually recursive processes

$$\mathsf{wAND}_0 \stackrel{def}{=} w_1.\mathsf{wAND}_1 + w_0{:}w_1.\mathsf{wAND}_0 + r_0{:}\{w_0, w_1\}.\mathsf{wAND}_0$$
$$\mathsf{wAND}_1 \stackrel{def}{=} w_1.\mathsf{wAND}_1 + w_0{:}w_1.\mathsf{wAND}_0 + r_1{:}\{w_0, w_1\}.\mathsf{wAND}_1$$

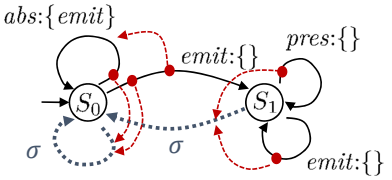
The process wAND_v represents the memory cell in a state where it has stored the value $v \in \{0, 1\}$. In either state, it offers to synchronise with a write action w_u to change its state to wAND_u, and also with a read action r_v to transmit its stored value v to a reader. The write actions w_u do not only change the state of the memory, they also block the read actions since they appear in their blocking sets $r_v{:}\{w_0, w_1\}$. Between the write actions, the blocking sets implement a precedence of w_1 over w_0. Notice that the blocking sets are visualised as dashed arrows.



▸ **Example 11** (Esterel Signals). Concurrent Esterel threads communicate via *signals* [4, 32]. A (pure, temporary) signal can have two statuses, *present* and *absent*. At the beginning of a synchronous macro-step, each signal is initialised to be absent, by default. The signal becomes present as soon as some thread emits it. It remains present thereafter for throughout the current macro-step, i.e., until the next clock cycle is started. The clock is a global synchronisation that taken by all threads simultaneously. The value of the signal is broadcast to all concurrent threads which can test it and branch their control-flow according to the signal's status. To maintain coherency of data-flow and deterministic behaviour, the signal status can only be read when no writing is possible any more. Esterel compilers conduct a causality analysis on the program and obtain a suitable static schedule when they generate imperative code. In Esterel hardware compilers, the scheduling is achieved dynamically by propagating signal statuses. Programs that cannot be scheduled to satisfy the emit-before-test protocol, or hardware that that does not stabilise is called *non-constructive* and rejected by the compiler. A pure, temporary signal is modelled by the following recursive behaviour:

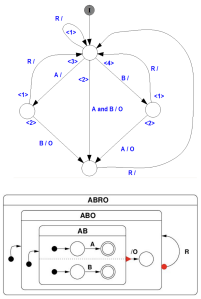
$$S_0 \stackrel{def}{=} abs{:}emit.S_0 + emit.S_1 + \sigma{:}\{abs, emit\}.S_0$$
$$S_1 \stackrel{def}{=} pres.S_1 + emit.S_1 + \sigma{:}\{pres, emit\}.S_0$$

which is depicted on side and structurally coherent according to Def. 16.



▸ **Example 12** (ABRO). The *hallo word* in synchronous languages, see [4], ABRO 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. [32]*

The behaviour of ABRO is expressed as a Mealy machine and as a SyncCharts, as displayed on the right. 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.



$$
\begin{array}{rcl}
\mathsf{ABRO} & \stackrel{def}{=} & \sigma.(\mathsf{R} \,|\, \mathsf{ABO}) \\
\mathsf{ABO} & \stackrel{def}{=} & (\mathsf{A} \,|\, \mathsf{B} \,|\, \mathsf{T} \,|\, \mathsf{O}) \setminus \{s,t\} \\
\mathsf{R} & \stackrel{def}{=} & r{:}r.\overline{k}_1{:}\overline{k}_1.\overline{k}_2{:}\overline{k}_2.\overline{k}_4{:}\overline{k}_4.\overline{k}_5{:}\overline{k}_5.\mathsf{ABRO} + \tau{:}r.\sigma.\mathsf{R} \\
\mathsf{A} & \stackrel{def}{=} & k_1{:}k_1.1 + a{:}\{k_1,a\}.\overline{s}{:}\overline{s}.1 + \sigma{:}\{k_1,a\}.\mathsf{A} \\
\mathsf{B} & \stackrel{def}{=} & k_2{:}k_2.1 + b{:}\{k_2,b\}.\overline{s}{:}\overline{s}.1 + \sigma{:}\{k_2,b\}.\mathsf{B} \\
\mathsf{T} & \stackrel{def}{=} & k_4{:}k_4.1 + s{:}\{k_4\}.\mathsf{T} + \overline{t}{:}\{k_4,\overline{t},s\}.1 + \sigma{:}\{k_4,\overline{t},s\}.\mathsf{T} \\
\mathsf{O} & \stackrel{def}{=} & k_5{:}k_5.1 + t{:}\{k_5,t\}.\overline{o}{:}\overline{o}.1 + \sigma{:}\{k_5,t\}.\mathsf{O}.
\end{array}
$$

Figure 4 ABRO.

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

In CCS^{spt} we represent the micro-steps as sequences of τ-transitions, i.e., local and asynchronous rendez-vous synchronisations, whereas the macro-step is captured as a global clock synchronisation. Unfortunately, the Mealy machine formalism is not scalable because of 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, as displayed on the right. The loop `loop P each R` 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 expressed in the blocking sets of CCS^{spt}. As such, we may present in the figure the following coding of the ABRO process in CCS^{spt}. The CCS^{spt} code follows the structure of the Esterel code. ABRO is obtained as a parallel composition $\mathsf{R} \,|\, \mathsf{ABO}$ combining a watchdog R with the behaviour ABO. The R is responsible to implement the reset loop at the superstate called ABO, as in the state charts. The ABO is the behaviour of the interior of this superstate, which corresponds to the statement `(await A || await B); emit O` in Esterel: it is the parallel composition of $\mathsf{A} \,|\, \mathsf{B}$ making up the behaviour of the region called AB with code `await A || await B`. The processes $\mathsf{T} \,|\, \mathsf{O}$ implement the synchronisation that waits for termination of AB to start the output process O. We assume that the channels a, b, r, o modelling interface signals `A`, `B`, `R`, `O` all have a unique sender and receiver (full-multicast in App. B.39). This is seen in our coding from the fact that the prefixes for these channels are self-blocking.

4 Constructive Scheduling for CCS^{spt}

This section presents the core elements of our scheduling theory for CCS^{spt} based on the notions of constructive enabling, coherence, policies, and policy-coherent processes. As indicated by the examples of Sec. 3, strong enabling generates confluent reductions in cases where rendez-vous actions have a unique sender and a unique receiver process. For multi-cast communication we need an even stronger notion of enabledness, called *constructive enabling*.

▸ **Definition 13** (Constructive Enabling). *The set* $iA^*(P) \subseteq \mathcal{L}$ *of* potential actions *is the smallest extension* $iA(P) \cap \mathcal{L} \subseteq iA^*(P)$ *such that if* $\ell \in iA^*(Q)$ *and* $P \xrightarrow{\alpha} Q$ *for* $\alpha \in \mathcal{R} \cup \{\tau\}$,

then $\ell \in \mathit{iA}^*(P)$. *A transition* $P \xrightarrow[R]{\alpha}_H Q$ *is called* constructively enabled, *or* c-enabled *for short, if* $H \cap (\overline{\mathit{iA}}^*(R) \cup \{\tau\}) = \{\,\}$.

Def. 13 models the deterministic semantics of synchronous programming languages like Esterel [3] which permit local multicast synchronisation. Associated with it is the notion of *Coherence* (Def. 16) which is inspired by Milner's *confluence* in CCS. The latter is preserved by prefixing, parallel composition and restriction only subject to serious restrictions identified in [22]. The former satisfies stronger closure properties and covers a larger class of processes.

▸ **Definition 14.** *Two c-actions* $\alpha_1{:}H_1[E_1]$ *and* $\alpha_2{:}H_2[E_2]$ *are called* interference-free *if the following holds for all* $i \in \{1,2\}$*:*
(1) *If* $\alpha_1 \neq \alpha_2$ *then* $\alpha_i \notin H_{3-i}$.
(2) *If* $\alpha_{3-i} = \tau$ *then* $H_i \cap (\overline{\mathit{iA}}^*(E_i) \cup \{\tau\}) = \{\,\}$.

Def. 14(1) says that distinct interference-free c-actions must not block each other. This ensures that they are not part of a conflicting choice in which each preempts the other. By Def. 14(2), non-interference with a silent action requires that the action does not get blocked by computations in its own reduction context.

▸ **Proposition 15.** *Suppose* $\alpha_1{:}H_1[E_1 \mid R]$ *and* $\alpha_2{:}H_2[E_2 \mid R]$ *are interference-free. Then,* $\alpha_1{:}H_1'[E_1]$ *and* $\alpha_2{:}H_2'[E_2]$ *are interference-free for arbitrary subsets* $H_1' \subseteq H_1$ *and* $H_2' \subseteq H_2$.

Coherence is formulated using an auxiliary form of transition $P \overset{\alpha}{\rightsquigarrow} Q$ which abbreviates a form of *residual* transition with the following two cases: (i) If $\alpha \in \mathcal{L}$ then $P \xrightarrow{\alpha} Q$ or $P \equiv Q$ and (ii) if $\alpha = \tau$ then $P \xrightarrow{\tau} Q$, $P \xrightarrow{\ell} Q$, or $P \xrightarrow{\overline{\ell}} Q$ for some $\ell \in \mathcal{R}$. This says that there is a transition from P to Q with an action that is a "factor" of α.

▸ **Definition 16** (Coherence). *A process* P *is* (structurally) coherent *if for all its derivatives* Q *the following holds, where* $\alpha_1, \alpha_2 \in \mathit{Act}$*: Given two transitions* $Q \xrightarrow[E_1]{\alpha_1}_{H_1} Q_1$ *and* $Q \xrightarrow[E_2]{\alpha_2}_{H_2} Q_2$ *such that* $\alpha_1 \neq \alpha_2$ *or* $Q_1 \not\equiv Q_2$ *or* $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ *and* $\alpha_i \notin H_{3-i}$. *Further, suppose the c-actions* $\alpha_i{:}H_i[E_i]$ *are interference-free. Then, there exist* H_i' *and processes* E_i', Q' *such that* $Q_1 \xrightarrow[E_2']{\alpha_2}_{H_2'} Q'$ *and* $Q_2 \xrightarrow[E_1']{\alpha_1}_{H_1'} Q'$ *with* $H_i' \subseteq H_i$ *and* $E_1 \overset{\alpha_2}{\rightsquigarrow} E_1'$ *and* $E_2 \overset{\alpha_1}{\rightsquigarrow} E_2'$. *In particular,* $E_1 \xrightarrow{\alpha_2} E_1'$ *and* $E_2 \xrightarrow{\alpha_1} E_2'$ *if* $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ *and* $\alpha_1 \neq \alpha_2$ *or* $Q_1 \not\equiv Q_2$, *or* $\{\alpha_1, \alpha_2\} \cap \mathcal{C} \neq \{\,\}$.

▸ **Theorem 17** (Coherence implies confluence). *Every coherent process is structurally confluent for c-enabled reductions.*

A coherent process cannot simultaneously offer both a clock and another distinct action without putting them in precedence order. Moreover, whenever a clock is enabled, then every further reduction of the process is blocked by the clock.

▸ **Proposition 18.** *If a coherent process* P *offers enabled transitions on a clock* σ *and another distinct action* $\alpha \neq \sigma$ *with* $P \xrightarrow[R_1]{\alpha}_{H_1} P_1$ *and* $P \xrightarrow[R_2]{\sigma}_{H_2} P_2$ *then* $\alpha \in H_2$ *or* $\sigma \in H_1$. *In particular, if* $\alpha = \tau$ *then* $\sigma \in H_1$.

Coherent processes are structurally deterministic for clock transitions. They do not offer a choice on the same visible label to structurally different successor states unless these actions also occur in concurrent threads.

▸ **Proposition 19** (Action Determinism). *Let* P *be coherent with* $P \xrightarrow[R_1]{\ell}_{H_1} P_1$ *such that* $\ell \in \mathcal{L}$ *and* $\ell \notin \mathit{iA}(R_1)$. *Then, for every transition* $P \xrightarrow[R_2]{\ell}_{H_2} P_2$ *we have* $P_1 \equiv P_2$. *In particular,* P *is structurally clock deterministic, i.e., if* $P \xrightarrow[R_1]{\sigma}_{H_1} P_1$ *and* $P \xrightarrow[R_2]{\sigma}_{H_2} P_2$ *then* $P_1 \equiv P_2$.

The problem of causality cycles that occurs in the parallel composition of two single-threaded processes stems from the symmetry between input and output actions. The standard trick to eliminate the problem as used e.g., in Occam, is to introduce a *causality order* so that outputs (sender actions) are deterministic by construction. All remaining choices are choices between inputs and these are resolved by the environment selecting the matching output. To express such causality properties of processes by static interfaces we refine the notion of a *sort* known from CCS for $\mathsf{CCS}^{\mathsf{spt}}$.

▸ **Definition 20** (Policy). *A* precedence policy *is a pair* $\pi = (L, \dashrightarrow)$ *where* $L \subseteq \mathcal{L}$ *is a subset of visible actions, called the* alphabet *of* π *and* $\dashrightarrow \subseteq L \times L$ *a binary relation on* L*, called the* precedence relation*. If* $(\ell_1, \ell_2) \in \dashrightarrow$ *we write* $\pi \Vdash \ell_1 \dashrightarrow \ell_2$*, or simply* $\ell_1 \dashrightarrow \ell_2$ *when* π *is understood. When* $\ell \in L$ *we say that* ℓ *is* admissible *under* π *and simply write* $\ell \in \pi$*. If* $\ell_1 \dashrightarrow \ell_2$ *we say that* ℓ_1 *takes* precedence *over* ℓ_2 *under* π*. If* $\ell_1, \ell_2 \in L$ *and* $\pi \nVdash \ell_1 \dashrightarrow \ell_2$ *as well as* $\pi \nVdash \ell_2 \dashrightarrow \ell_1$ *we say that* ℓ_1 *and* ℓ_2 *are* concurrently independent *under* π*, written* $\pi \Vdash \ell_1 \diamond \ell_2$ *or simply* $\ell_1 \diamond \ell_2$ *when* π *is understood.*

We obtain a partial ordering $\pi_1 \preceq \pi_2$ on precedence policies by subset inclusion. Specifically, we have $(L_1, \dashrightarrow_1) \preceq (L_2, \dashrightarrow_2)$ if $L_1 \subseteq L_2$ such that for all $\ell_1, \ell_2 \in L_1$, if $\ell_1 \dashrightarrow_2 \ell_2$ then $\ell_1 \dashrightarrow_1 \ell_2$. Intuitively, if $\pi_1 \preceq \pi_2$ then π_2 is a tightening of π_1 in the sense that it exports more labels (resources) subject to possibly fewer precedences (causality constraints) than π_1. There is also the normal inclusion ordering $\pi_1 \subseteq \pi_2$ between policies defined by $L_1 \subseteq L_2$ and $\dashrightarrow_1 \subseteq \dashrightarrow_2$. It differs from $\preceq$ in the inclusion direction of the precedence alphabets. Another notation we need is restriction. For any policy $\pi = (L, \dashrightarrow)$ and set of labels $K \subseteq \mathcal{L}$ we denote by $\pi \setminus K$ the policy π restricted to the alphabet $L - (K \cup \overline{K})$ in the obvious way.

For any policy π on alphabet L we define its *dual* as a policy $\overline{\pi}$ on the set of labels $\overline{L}$ where for all $\overline{\ell}_1, \overline{\ell}_2 \in \overline{L}$ we stipulate $\overline{\pi} \Vdash \overline{\ell}_1 \dashrightarrow \overline{\ell}_2$ iff $\ell_1 = \ell_2$ or $\pi \Vdash \ell_1 \diamond \ell_2$, i.e., both $\pi \nVdash \ell_1 \dashrightarrow \ell_2$ and $\pi \nVdash \ell_2 \dashrightarrow \ell_1$. Intuitively, the dual policy $\overline{\pi}$ consists of matching copies $\overline{\ell}$ of all labels ℓ from π and puts them in a precedence relation precisely if their original copies are identical or do not stand in precedence to each other, in any direction. This generates a form of "negation" $\overline{\pi}$ from π, which is reflexive and symmetric. Note that $\overline{\overline{\pi}}$ need not be identical to π but contains more precedences. In general, we have $\overline{\overline{\pi}} \preceq \pi$ and $\overline{\overline{\pi}} = \pi$ if π is symmetric. The policy $\overline{\pi}$ captures the independences between actions of π and is called the associated *independence alphabet*, in the sense of trace theory [9]. Analogously, $\overline{\overline{\pi}}$ expresses the dependencies of actions in π, called the associated *dependence alphabet*.

We are going to use policies to measure the degree of concurrency exhibited by a process. The alphabet of the $\preceq$-maximal policy π_{max} contains all labels $\mathcal{L}$ and its precedence relation $\dashrightarrow_{max}$ is empty. It is the largest policy in the $\preceq$ ordering. Its dual $\overline{\pi}_{max}$ contains all labels and precedences between all of them, i.e., $\overline{\pi}_{max} \nVdash \ell_1 \dashrightarrow \ell_2$ for all $\ell_1, \ell_2 \in \mathcal{L}$. The $\preceq$-minimal policy π_{min} has empty alphabet and empty precedences. It is self-dual, i.e., $\overline{\pi}_{min} = \pi_{min}$. The following definition explains how policies can statically specify the externally observable scheduling behaviour of a process. The maximal policy π_{max} is least restrictive in the sense that it permits a process to offer arbitrary actions with arbitrary blocking. The minimal policy π_{min} is most restrictive in that it does not permit any externally visible actions.

▸ **Definition 21** (Conformance). P conforms *to a policy* π *if for each of its derivatives* Q*: If* $Q \xrightarrow[R]{\ell}_H Q'$ *with* $\ell \in \mathcal{L}$*, then* $\ell \in \pi$ *and for all* $\ell' \in \mathcal{L}$*, if* $\ell' \in H$ *then* $\pi \Vdash \ell' \dashrightarrow \ell$*.*

If P conforms to π then the admissible actions of π corresponds to "sorts" of P in CCS. It gives an upper bound on the labels that can be used by the process. The precedence relation $\dashrightarrow$ is specific to $\mathsf{CCS}^{\mathsf{spt}}$. It provides for a static over-approximation of the blocking between

visible actions, i.e., which labels can be in a choice conflict with each other. Specifically, if $\pi \Vdash \ell_1 \diamond \ell_2$ then all transitions with labels ℓ_1 and ℓ_2 must be from concurrent threads and thus cannot block each other. If $\pi \Vdash \ell_1 \dashrightarrow \ell_2$ and $\ell_1 \neq \ell_2$ are distinct, then P may take ℓ_1 and ℓ_2 transitions to completely different states from the same thread. Necessarily, these must be transitions combined in a prioritised sum, where one of ℓ_i blocks the other ℓ_{3-i}. A special case occurs if $\ell_1 = \ell_2$. Then, the presence of $\pi \Vdash \ell \dashrightarrow \ell$ permits P and its derivatives Q to execute ℓ without any guarantee that ℓ can synchronise with arbitrarily many concurrent partners. In contrast, if the policy specifies $\pi \Vdash \ell \diamond \ell$, then a conformant P can execute ℓ only in concurrent threads and each ℓ-transition is arbitrarily repeatable.

Here we are interested in a restricted class of processes that conform to policies with special properties. We write $P : \pi$ to express that P is coherent and conforms to policy π.

▸ **Definition 22** (Pivotable, Precedence-closed, Input-scheduled Processes)**.**

- *A policy π is a* pivot *policy if $\overline{\pi} \preceq \pi$.*
- *A policy π is* input-scheduled *if $\pi \subseteq \pi_{is}$ where $\pi_{is} = (L_{is}, \dashrightarrow_{is})$ given by $L_{is} = \mathcal{L}$ and $\pi_{is} \Vdash \ell_1 \dashrightarrow \ell_2$ iff $\ell_1 = \ell_2$ or $\ell_1, \ell_2 \in \mathcal{A} \cup \mathcal{C}$.*
- *A policy π is* precedence-closed *for $L \subseteq \mathcal{L}$ if $\ell_1 \in L$ and $\pi \Vdash \ell_1 \dashrightarrow \ell_2$ implies $\ell_2 \in L$.*

A process is pivotable/input-scheduled/precedence-closed *if $P : \pi$ where π is a pivot policy/input-scheduled/precedence-closed.*

In a pivotable process, the choice between any two synchronisations is resolved by the precedences in third independent thread, acting as the "pivot". For instance, the process $P \stackrel{def}{=} a + \overline{b}{:}a \,|\, b + \overline{a}{:}b$ is not pivotable (see also Ex. 8 in App. B). It consists of two threads, each able to synchronise on a and also b. Which one is chosen, must be resolved between the two threads. This is not possible, because the first wants channel a to take priority over $\overline{b}$ and the second the other way around. This is reflected in any associated policy π for which P is conformant, which must satisfy $\pi \Vdash a \dashrightarrow \overline{b}$ and $\pi \Vdash b \dashrightarrow \overline{a}$. Obviously, this is not a pivot policy which requires $\pi \Vdash a \diamond \overline{b}$ or $\pi \Vdash \overline{a} \diamond b$. For contrast, the process $Q \stackrel{def}{=} a + \overline{b}{:}a \,|\, b \,|\, \overline{a}$ is pivotable as it conforms to the pivot policy with $\pi \Vdash a \dashrightarrow \overline{b}$ and $\pi \Vdash \overline{a} \diamond b$. Here, the choice between synchronising on a or on b is uniquely resolved by the first thread $a + \overline{b}{:}a$ which, being offered $\overline{a}$ and b concurrently by its environment, takes the synchronisation $\tau = a \,|\, \overline{a}$.

Input-scheduled processesare processes in which choices between distinct actions need to be resolved only between receiver actions and clocks. In other words, each thread executes its send actions always deterministically, rather than in a preemption context. The policy π_{is} forces all output actions to be concurrently independent from each other. On the other hand, the policy permits a process to introduce arbitrary priorites between receiver actions and clocks. Note that π_{is} is a pivot policy and thus every input-scheduled process is pivotable.

Reductions of pivotable processes never block because of two threads entering into a circular precedence deadlock with each other. Therefore, it suffices to check enabling relative to the concurrent environment in which a rendez-vous synchronisation is happening. One can show that for pivotable processes of at most two threads, the three operational semantics of CCS (admissible), of $\mathsf{CCS}^{\mathsf{Ph}}$ (weak enabling) and our $\mathsf{CCS}^{\mathsf{spt}}$ (strong enabling) all coincide. This is a consequence of the fact (cf. Lem. 45 and 46 in the Appendix) that the side condition in the rule (Par_3) is never introducing the silent action τ into the blocking sets. In general, pivotable processes may consist of three or more interacting threads. Then, blocking may occur and the semantics of CCS, $\mathsf{CCS}^{\mathsf{Ph}}$ and $\mathsf{CCS}^{\mathsf{spt}}$ are different. However, two single-threaded pivotable processes never block each other.

▸ **Lemma 23.** *Let $P_1 : \pi$ and $P_2 : \pi$ be coherent for the same pivot policy π. If P_1 and P_2 are single-threaded, then every admissible transition of $P_1 \,|\, P_2$ is c-enabled.*

The following proposition says that a pivotable process cannot offer a clock action and exhibit another clock or a rendez-vous synchronisation at the same time. This means that in this class of processes, clocks and reductions are sequentially scheduled.

▸ **Proposition 24** (Clock Maximal Progress). *Suppose P is coherent and pivotable and $\sigma \in \mathit{iA}(P)$. Then, for all $\ell \in \mathcal{L}$ with $\ell \neq \sigma$ we have $\ell \notin \mathit{iA}(P)$ or $\overline{\ell} \notin \mathit{iA}(P)$. In particular, P is in normal form, i.e., there is no reduction $P \xrightarrow{\tau} P'$.*

Prop. 24 can be seen as saying that in pivotable processes all clocks take lowest precedence. As a result, clock transitions satisfy *maximal progress*, i.e., a clock is only enabled on normal forms, i.e., if there is no reduction possible. Thus clocks behave like time steps in the standard timed process algebras [15, 8]. More importantly (Thm. 16) all reductions are confluent. So, a pivotable process, no matter in which order the reductions are executed, when it terminates, it computes a unique normal form. At this point it either stops if there is no clock possible, or it offers a clock step (pausing) to a continuation process from which a new normal form is produced. In this way, coherent pivotable processes correspond to synchronous streams.

It remains to be seen how we can obtain coherent processes systematically by construction. We identify a set of closure operators for coherent processes based on policy-coherence. The associated Propositions 47–57 (Appendix) cover idling and prefixes (SSec. D.1), summation (SSec. D.2), parallel composition (SSec. D.3), repetition (SSec. D.4), hiding (SSec. D.5) and restriction (SSec. D.6). For lack of space, the results are summarised:

- $0 : \pi$ for all π.
- If $Q : \pi$ then for every $\ell \in \mathcal{R}$ we have $\ell{:}H.Q : \pi$ if $\ell \in H \subseteq \{\ell' \mid \pi \Vdash \ell' \dashrightarrow \ell\}$.
- If $Q : \pi$ and clock $\sigma \in \mathcal{C}$, then $\sigma{:}H.Q : \pi$ if $H \subseteq \{\beta \mid \pi \Vdash \beta \dashrightarrow \sigma\}$.
- If $P_1 : \pi_1$ and $P_2 : \pi_2$ and for all pairs of initial transitions $P \xrightarrow[F_1]{\alpha_1}_{H_1} P_1$ and $P \xrightarrow[F_2]{\alpha_2}_{H_2} P_2$ we have $\alpha_1 \neq \alpha_2$ as well as $\alpha_1 \in H_2$ or $\alpha_2 \in H_1$. Then $P_1 + P_2 : \pi$ if $\pi_1 \subseteq \pi$ and $\pi_2 \subseteq \pi$.
- If $Q : \pi$ and $R : \pi$ where $\pi \setminus \mathcal{R}$ is a pivot policy. Then $(Q \,|\, R) : \pi$.
- If $P : \pi$ and $\pi \setminus \mathcal{R}$ is a pivot policy then for each $\ell \in \pi \setminus \mathcal{R}$, the sequential bang prefix $\ell^*{:}H.P : \pi$ if $H \subseteq \{\ell' \mid \pi \Vdash \ell' \dashrightarrow \ell\}$.
- If $Q : \pi$ and $\pi \not\Vdash \sigma \dashrightarrow \ell$ for all $\sigma \in L$, then $Q \,/\, L : \pi$.
- If $Q : \pi$ and π precedence-closed for $L \cup \overline{L}$, then $(Q \setminus L) : \pi - L$.

These are processes that are structurally deterministic for clocks (Prop. 19) and structurally confluent for strongly enabled reductions (Thm. 17).

5 Further work

We list in spare order some possible future works:

- C-actions are somewhat related with continuation passing style, and, in some sense they are more powerful because they do a kind of "forecast analysis" of all possible future actions: studying this relation, possibly in relation with abstract interpretation techniques, would be worth of study.
- Clocks are unordered: introducing a partial order could be an interesting, although economic and fine grained, way to reason about time and time priorities.
- Plugging a type system to $\mathsf{CCS}^{\mathsf{spt}}$ seems worth of study: among possible type disciplines we just mention: behavioural type systems, including session-types to capture rendez-vous interactions; intersection-types, including non-idempotent ones to characterise termination and resources awareness, and linear-types to model consuming resources.

References

1. H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, revised edition, 1984.
2. G. Barrett. The semantics of priority and fairness in OCCAM. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Proc. MFPS'89*, 1989.
3. G. Berry. *The Constructive Semantics of Pure Esterel*. Draft Book, 1999. `ftp://ftp-sop.inria.fr/esterel/pub/papers/constructiveness3.ps`.
4. G. Berry. The Foundations of Esterel. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 425–454. MIT Press, 2000.
5. G. Berry and G. Boudol. The chemical abstract machine. *Theor. Comput. Sci.*, 96(1):217–248, 1992. URL: `https://doi.org/10.1016/0304-3975(92)90185-I`, `doi:10.1016/0304-3975(92)90185-I`.
6. R. Bocchino, V. Adve, S. Adve, and M. Snir. Parallel programming must be deterministic by default. In *Proceedings of the First USENIX conference on Hot topics in parallelism*, pages 4–4, 2009.
7. J. Camilleri and G. Winskel. CCS with priority choice. *Information and Computation*, 116(1):26–37, 1995.
8. R. Cleaveland, G. Lüttgen, and M. Mendler. An algebraic theory of multiple clocks. In A. Mazurkiewicz and J. Winkowski, editors, *Proceedings of the International Conference on Concurrency Theory CONCUR'97*, pages 166–180. Springer, 1997. LNCS 1243.
9. V. Dieckert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.
10. S. A. Edwards. On determinism. In M. Lohstroh et. al., editor, *Lee Festschrift*, pages 240–253. Springer, 2018.
11. P. Le Guernic, T. Goutier, 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, September 1991.
12. M. Mendler H. R. Andersen. An asynchronous process algebra with multiple clocks. In D. Sannella, editor, *In Proc. ESOP'94*, pages 58–73, 1994.
13. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.
14. D. Harel, A. Pnueli, J. Pruzan-Schmidt, and R. Sherman. On the formal semantics of Statecharts. In *Logic in Computer Science (LICS'87)*, pages 54–64. IEEE Press, 1987.
15. M. Hennessy and T. Regan. A process algebra for timed system. *Information and Computation*, 117:221–239, 1995.
16. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985. 256 pages.
17. C. D. Kloos and P. Breuer, editors. *Formal Semantics for VHDL*. Kluwer, 1995.
18. J. W. Klop. *Combinatory reduction systems*. PhD thesis, Univ. Utrecht, 1980.
19. E. A. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.
20. E. A. Lee and T. M. Parks. Dataflow process networks. In *Proceedings of the IEEE*, pages 773–799, May 1995.
21. G. Lüttgen and M. Mendler. When 1 clock is not enough. In L. Aceto and A.D. Gordon, editors, *Algebraic Process Calculi: The First Twenty Five Years and Beyond (PA'05)*, number NS-05-03 in BRICS Notes Series, pages 155–158, June 2005.
22. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
23. R. Milner. Functions as processes. *Math. Struct. Comput. Sci.*, 2(2):119–141, 1992. URL: `https://doi.org/10.1017/S0960129500001407`, `doi:10.1017/S0960129500001407`.
24. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992. URL: `https://doi.org/10.1016/0890-5401(92)90008-4`, `doi:10.1016/0890-5401(92)90008-4`.
25. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, II. *Inf. Comput.*, 100(1):41–77, 1992. URL: `https://doi.org/10.1016/0890-5401(92)90009-5`, `doi:10.1016/0890-5401(92)90009-5`.

26 B. Norton, G. Luettgen, and M. Mendler. A compositional semantic theory for component-based synchronous design. In R. Amadio and D. Lugiez, editors, *Int'l Conf. on Concurrency Theory (CONCUR'2003)*, pages 461–476, Marseille, September 2003. Springer LNCS 2761.

27 I. Phillips. CCS with priority guards. In K. G. Larsen and M. Nielsen, editors, *Proc. Concur 2001*, pages 305–320, 2001.

28 G. D. Plotkin. The origins of structural operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:3–15, 2004. URL: `https://doi.org/10.1016/j.jlap.2004.03.009`, `doi:10.1016/J.JLAP.2004.03.009`.

29 G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:17–139, 2004.

30 V. Natarajan R. Cleaveland, G. Lüettgen. *Handbook of Process Algebra*, chapter Priority in Process Algebra. Elsevier, 2001.

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

32 Esterel Technologies. The Esterel v7 reference manual version v7_30 – initial IEEE standardization proposal. Technical report, Esterel Technologies, 679 av. Dr. J. Lefebvre, 06270 Villeneuve-Loubet, France, November 2005.

33 S.H. Unger. Asynchronous sequential switching circuits with unrestricted input changes. *IEEE Transactions on Computers*, C-20(12):1437–1444, 1971. `doi:10.1109/T-C.1971.223155`.

34 L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990. URL: `https://doi.org/10.1145/79173.79181`, `doi:10.1145/79173.79181`.

35 R. J. van Glabbeek. Notes on the methodology of CCS and CSP. *Theor. Comput. Sci.*, 177(2):329–349, 1997. URL: `https://doi.org/10.1016/S0304-3975(96)00251-4`, `doi:10.1016/S0304-3975(96)00251-4`.

36 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. PLDI'14*, Edinburgh, UK, June 2014. ACM.

A From Asynchronous Weak to Synchronous Strong Priorities

This rather informal section give to the curious reader a nice view of our personal pilgrimage of previous work looking at reaching more control and determinism in asynchronous process algebra. This trip lead us to consider adding clocks to the previous literature and reach confluence in a synchronous setting.

A.1 CCS with (Weak) Priorities

Let us look in more detail on how the precedence mechanism suggested above might be derived from the classical priorities studied for CCS. As a simple example, suppose S is a (synchronous write, asynchronous read) memory cell that can be read at any time offering the read action r, but only be written once with the write action w. Ignoring data values, the cell would be modelled in plain CCS[2] by the equations $S \stackrel{def}{=} w.S' + r.S''$ where $S' \stackrel{def}{=} r.S'$ and $S'' \stackrel{def}{=} r.S'' + e.S''$. Initially, memory cell S is empty and offers a choice between a read action r and a write action w. If the write action is performed, the cell receives a value and changes to state S'. In this filled state S' only read actions are admissible to access the stored value,

[2] Here we use equations systems to define recursive processes. The process S in solved form is $S \stackrel{def}{=} w.(\mathsf{rec}\, x.\, r.x) + r.(\mathsf{rec}\, x.\, (r.x + e.x))$, where rec is a recursion operator such as used in CCS. In the formal syntax that we use later, we will not use recursion but repetition to express recursive processes. This will turn out to be more convenient to control confluence, as we shall see.

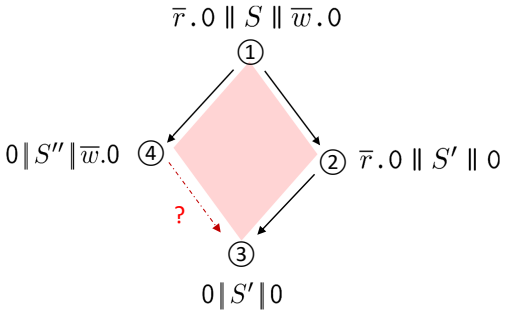


Figure 5 Symmetric Choice. The final outcome, in states ③ and ④, is non-determinate because it depends on the order of read and write. The continuations states are $S' \stackrel{def}{=} r.S'$ and $S'' \stackrel{def}{=} r.S'' + e.S''$. For confluence, the error state ③ should permit a (delayed) read to move forward and (eventually) meet with state ④, indicated by the dashed red arrow. However, this is not in general guaranteed.

the state does not change. When the read action is executed on the empty cell S, however, the continuation state is S'' which represents an error state. In the error state S'' only a read or an error action e is possible, but no write and the cell remains in error. Observe that only the read action is always admissible in both states S and S', while the write only in state S. If S is put in parallel with a reader offering $\overline{r}$ and a writer offering $\overline{w}$, then the final state of S depends on the order between the $\overline{r}$ and $\overline{w}$ actions. If we synchronise with the external sequence $\overline{r}.\overline{w}$ on S where the read action is first, the writer gets blocked on the error state S'', while if we exercise $\overline{w}.\overline{r}$ where the write action is first, both can proceed and we obtain filled state S' (in term rewriting system jargon [18] this is called "critical pairs").

Formally, let $R \stackrel{def}{=} \overline{r}.0$ be a reader process with a single read and $W \stackrel{def}{=} \overline{w}.0$ the writer with a single write, both terminating in the empty behaviour 0. Then, according to the operational semantics of CCS, we have the two execution sequences $R \mid S \mid W \xrightarrow{\tau} 0 \mid S'' \mid W$ and $R \mid S \mid W \xrightarrow{\tau} R \mid S' \mid 0 \xrightarrow{\tau} 0 \mid S' \mid 0$, where both configurations $0 \mid S'' \mid W$ and $0 \mid S'' \mid 0$ are distinct normal forms, in the sense that no reductions are possible anymore. This is pictorially illustrated in Fig. 5. Observe that the behaviour is not only *non-deterministic* in the sense that there is more than one scheduling sequence. It is also non-deterministic as the final outcome depends on the scheduling decisions.

Among many CCS extension with priorities, we looked the one of Camilleri and Winskell [7], referred to as $\mathsf{CCS}^{\mathsf{CW}}$ in the sequel. $\mathsf{CCS}^{\mathsf{CW}}$ provide a *prioritised choice* operator $Q \stackrel{def}{=} \alpha_1.Q_1 \triangleright \alpha_2.Q_2$ called "prisum" that gives the action α_1 priority over α_2 and blocks the second prefix α_2 in case $\alpha_1 = \alpha_2$. Such priorities help us resolve the issue. The problem with non-determinism originates from the preemptive behaviour of the write-read choice $w.S' + r.S''$. It permits the read and the write actions to compete with each other for the next state of the process S. In data-flow and functional programming, we resolve the conflict by making the write action take precedence over the read one. First, the data-flow variable must be written by the *producer* process, then it can be read by the *consumers*, if any. This "write-before-read scheduling policy" is a *prioritised choice*, represented in prioritised $\mathsf{CCS}^{\mathsf{CW}}$ [7] as $S \stackrel{def}{=} w.S' \triangleright r.S''$ in which the prisum $\triangleright$ operator enforces a precedence from left to right. The store S is willing to engage in a read action r (to the right of $\triangleright$) only if the environment does not offer a write w (on the left of $\triangleright$) at the same time, concurrently.

Another important study in the endeavour of adding priorities to CCS is the work of Phillips. In Phillips' extension of CCS with priority guards [27], referred to as $\mathsf{CCS}^{\mathsf{Ph}}$, the previous behavior is expressed by $S \stackrel{def}{=} \{\}{:}w.\, S' + \{w\}{:}r.\, S''$, where each action is pre-annotated

by a set of actions that take priority. Process S will synchronise with the reader process R to move to the error state S'' with *guarded* action $\{w\}{:}r$. The *priority guard* $\{w\}$ states that this can only happen if the environment does not offer a write w. Otherwise, the only action available is $\{\}{:}w$ which makes the cell move into state S'. The write action has an empty guard and so cannot be blocked. If both $\overline{w}$ and $\overline{r}$ are offered concurrently, then only the higher-prioritised write action $\overline{w}$ will go ahead.

The standard way to implement this scheduling in prioritised CCS is via a transition relation that keeps track of blocking information. In line with the approaches of [7, 27, 30], let us write $P \xrightarrow{\alpha}_H P'$ to denote that process P can engage in an action α to continue as P', provided the concurrent environment does not offer any action from H. We will call H the *blocking set* of the transition. In Plotkin's SOS, the rules for prefix and choice in $\mathsf{CCS}^{\mathsf{CW}}$ are the following[3]:

$$\frac{}{\alpha.P \xrightarrow{\alpha}_{\{\}} P}\ (Act) \qquad \frac{P \xrightarrow{\alpha}_H P'}{P \triangleright Q \xrightarrow{\alpha}_L P'}\ (PriSum_1) \qquad \frac{Q \xrightarrow{\alpha}_H Q' \qquad \tau, \alpha \notin \mathsf{iA}(P)}{P \triangleright Q \xrightarrow{\alpha}_{H \cup \mathsf{iA}(P)} Q'}\ (PriSum_2)$$

where $\mathsf{iA}(P)$ (defined formally later) represents the set of *initial actions* of P that are offered by P in some environment and $\triangleright$ is the priority sum defined by $\mathsf{CCS}^{\mathsf{CW}}$. From rule (Act), individual prefixes generate transitions without any blocking constraints. Rules $(PriSum_1)$ and $(PriSum_2)$ describe how the transitions of a choice $P \triangleright Q$ are obtained from the transitions of P and of Q: more in details, rule $(PriSum_1)$ lets all transitions of the higher-prioritised process P pass through to form a transition of the choice $P \triangleright Q$ without restriction. This is different for Q, where, according to rule $(PriSum_2)$, an action of the lower-prioritised process Q can only be executed, and thus P preempted by it, if P does not have an initial silent action and if it does not offer the same action α, already. All initial actions of P are added to the blocking set Q's transition. This will ensure that these get blocked in concurrent contexts in which P has a communication partner. The blocking sets take effect in the rules for parallel composition displayed below:

$$\frac{P_1 \xrightarrow{\alpha_1}_{H_1} P_1' \qquad H_1 \cap \overline{\mathsf{iA}}(P_2) = \{\}}{P_1 \,|\, P_2 \xrightarrow{\alpha_1}_{H} P_1' \,|\, P_2}\ (Par_1) \qquad \frac{P_2 \xrightarrow{\alpha_2}_{H_2} P_2' \qquad H_2 \cap \overline{\mathsf{iA}}(P_1) = \{\}}{P_1 \,|\, P_2 \xrightarrow{\alpha_2}_{H_2} P_1 \,|\, P_2'}\ (Par_2)$$

$$\frac{P_1 \xrightarrow{\alpha_1}_{H_1} P_1' \qquad P_2 \xrightarrow{\alpha_2}_{H_2} P_2' \qquad \alpha_1 = \overline{\alpha}_2 \qquad H_1 \cap \overline{\mathsf{iA}}(P_2) = \{\} \qquad H_2 \cap \overline{\mathsf{iA}}(P_1) = \{\}}{P_1 \,|\, P_2 \xrightarrow{\tau}_{H_1 \cup H_2} P_1' \,|\, P_2'}\ (Par_3)$$

More in details, rules (Par_i) for $i \in \{1, 2\}$ define an action α_i by $P_1 \,|\, P_2$ offered in one of the sub-processes P_i. Such is enabled if the competing concurrent process P_{3-i} has no initial action that is in the blocking set H_i of action α_i, i.e. when the condition $H_i \cap \overline{\mathsf{iA}}(P_{3-i}) = \{\}$ is satisfied. Rule (Par_3) implements the synchronisation between an action α_1 in P_1 and the associated matching action $\alpha_2 = \overline{\alpha}_1$, originating from P_2. The side-condition is the same: each process P_i only accepts the matching action call from P_{3-i} if it does not offer any action that blocks action α_i. Let us see how this blocking semantics works for our running example of a write-once memory cell, where we now enforce a write-before-read priority on

[3] For the fragment of $\mathsf{CCS}^{\mathsf{CW}}$ that we are considering, the precise connection is the following: $\vdash_R P \xrightarrow{\alpha} P'$ in $\mathsf{CCS}^{\mathsf{CW}}$ iff R is a subset of *out*-actions and there is a subset L of *in*-actions such that $P \xrightarrow{\alpha}_H P'$ and either α is an *out*-action and $H = \{\}$ or α is an *in*-action with $\overline{\alpha} \in R$ and $R \cap \overline{H} = \{\}$.

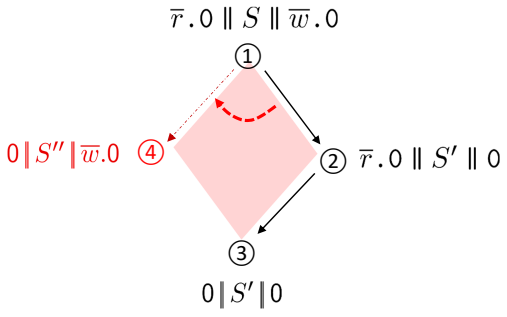


Figure 6 Prioritised Choice. The read-write data race on memory cell S resolved by scheduling priority. The write takes precedence over the read, indicated by the dashed arrow. This removes the top left transition labelled $\tau{:}\{w\}$. Since there is only one execution path, the system is deterministic. The continuations states $S' \stackrel{def}{=} r.S'$ and $S'' \stackrel{def}{=} r.S'' + e.S''$ of the cell are the same as in Fig. 5.

the empty cell S: $S \stackrel{def}{=} w.S' \triangleright r.S''$, $S' \stackrel{def}{=} r.S'$, and $S'' \stackrel{def}{=} r.S'' + e.S''$. The cell S has two potential transitions that can be fired, namely

$$\frac{\dfrac{}{w.S' \xrightarrow{w}_{\{\}} S'}\,(Act)}{S \xrightarrow{w}_{\{\}} S'}\,(PriSum_1) \qquad \text{or} \qquad \frac{\dfrac{}{r.S'' \xrightarrow{r}_{\{\}} S''}\,(Act) \quad \tau, r \notin \mathsf{iA}(w.S')}{S \xrightarrow{r}_{\{w\}} S''}\,(PriSum_2)$$

considering that $\mathsf{iA}(w.S') = \{w\}$. The latter transition synchronises with the reader process' $\overline{r}$ transition to give $R \mid S \xrightarrow{\tau}_{\{w\}} 0 \mid S''$ as verified by the derivation

$$\frac{\dfrac{}{R \xrightarrow{\overline{r}}_{\{\}} 0}\,(Act) \quad \dfrac{\vdots}{S \xrightarrow{r}_{\{w\}} S''} \quad \{\} \cap \overline{\mathsf{iA}}(S) = \{\} \quad \{w\} \cap \overline{\mathsf{iA}}(R) = \{\}}{R \mid S \xrightarrow{\tau}_{\{w\}} 0 \mid S''}\,(Par_3)$$

in which the side-conditions $\{w\} \cap \overline{\mathsf{iA}}(R) = \{w\} \cap \{r\} = \{\}$ and $\{\} \cap \overline{\mathsf{iA}}(S) = \{\}$ of rule (Par_3) are satisfied. This is fine since a single reader should be able to access the store, as long as there is no conflicting write. Note that the resulting silent transition is guarded by the write action w to record this constraint. Now if we add the writer process, too, then we get stuck. The writer's initial action $\overline{w} \in \mathsf{iA}(W)$ conflicts with the silent transition. The execution in the context $E = [\cdot] \mid W$ is blocked by rule (Par_1), because $\{w\} \cap \overline{\mathsf{iA}}(W) = \{w\} \cap \{w\} \neq \{\}$. The derivation tree

$$\frac{\dfrac{\dfrac{}{R \xrightarrow{\overline{r}}_{\{\}} 0}\,(Act) \quad \dfrac{\dfrac{}{r.S'' \xrightarrow{r}_{\{\}} S''}\,(Act)}{S \xrightarrow{r}_{\{w\}} S''}\,(PriSum_2)}{R \mid S \xrightarrow{\tau}_{\{w\}} 0 \mid S''}\,(Par_3) \qquad \{w\} \cap \mathsf{iA}(W) \neq \{\}}{R \mid S \mid W \not\xrightarrow{\tau}_{\{w\}} 0 \mid S'' \mid W}\,(Par_1)$$

where the side-conditions of $(PriSum_2)$ and of (Par_3) are satisfied, blocks at side-condition of rule (Par_1). On the other hand, the writing of S by W followed by the reading of S' by R can be derived, i.e., we have $R \mid S \mid W \xrightarrow{\tau}_{\{\}} R \mid S' \mid 0$ and $R \mid S' \mid 0 \xrightarrow{\tau}_{\{\}} 0 \mid S' \mid 0$. The derivation

trees in the SOS are as follows

$$\dfrac{\dfrac{\dfrac{\dfrac{}{w.S' \xrightarrow{w}_{\{\}} S'}\,(Act)}{S \xrightarrow{w}_{\{\}} S'}\,(PriSum_1)}{R \mid S \xrightarrow{w}_{\{\}} R \mid S'}\,(Par_2) \qquad \dfrac{}{W \xrightarrow{\overline{w}}_{\{\}} 0}\,(Act)}{R \mid S \mid W \xrightarrow{\tau}_{\{\}} R \mid S' \mid 0}\,(Par_3) \qquad \dfrac{\dfrac{}{R \xrightarrow{\overline{r}}_{\{\}} 0}\,(Act) \qquad \dfrac{\dfrac{}{S' \xrightarrow{r}_{\{\}} S'}\,(Act)}{S' \mid 0 \xrightarrow{r}_{\{\}} S' \mid 0}\,(Par_1)}{R \mid S' \mid 0 \xrightarrow{\tau}_{\{\}} 0 \mid S' \mid 0}\,(Par_3)$$

where the blocking side-conditions of (Par_1), (Par_2) and (Par_3) are all trivially satisfied. The scheduling which resolves the non-determinism is depicted in Fig. 6.

A.2 Asynchronous CCS with (Strong) Priorities is Confluent

Our example from the previous section shows how (weak) priority-based scheduling helps us to exercise control over the execution order in an asynchronous process calculus with rendezvous communication. The surprising observation is that very little needs to be changed in the traditional notion of priority for CCS to achieve confluence. For this we need to look how to "tune" of the classical weak priorities for our purposes. Our fix will make the difference between *(weak) priority-based* and *(strong) priority-based* scheduling (that we sometimes call "precedence-based" in this paper).

What we want is that every process using only strong priority satisfies the following *weak local confluence* property [18]. Suppose P admits two transitions to different states $P_1 \neq P_2$ that do not block each other, i.e., $P \xrightarrow{\alpha_1}_{H_1} P_1$ and $P \xrightarrow{\alpha_2}_{H_2} P_2$ such that $\alpha_1 \notin H_2$ and $\alpha_2 \notin H_1$. Then, each action α_i (for $i \in \{1,2\}$) is still admissible in P_{3-i}, and there exist P' and $H_i' \subseteq H_i$ such that $P_i \xrightarrow{\alpha_{3-i}}_{H_i'} P'$. The assumption $\alpha_i \notin H_{3-i}$ uses the blocking sets to express non-interference between transitions. Then, our weak[4] local confluence expresses a weak "diamond property" stating that every diverging choice of non-interfering transitions immediately reconverges. The subset inclusions $H_i' \subseteq H_i$ ensure that delaying an admissible action does not introduce more chances of interference.

A.2.1 Fix 1

The standard notion of priority for CCS, as implemented in $\mathsf{CCS}^{\mathsf{CW}}$ or $\mathsf{CCS}^{\mathsf{Ph}}$, of course was not designed to restrict CCS for determinism but to extend it conservatively to be able to express scheduling control. Even in the fragment of $\mathsf{CCS}^{\mathsf{CW}}$ where all choices are prioritised, weak confluence fails. For instance, although our memory cell S behaves deterministically for a single reader R and a single writer W, in the context $E \stackrel{def}{=} W_1 \mid [\cdot] \mid W_2$ of two writers $W_i \stackrel{def}{=} \overline{w}.W_i'$, it creates a race. The two writers compete for the single write action on S and only one can win. Using the operational semantics of $\mathsf{CCS}^{\mathsf{CW}}$ we can derive the (non-interfering) transitions $W_1 \mid S \mid W_2 \xrightarrow{\tau}_{\{\}} W_1' \mid S' \mid W_2$ and $W_1 \mid S \mid W_2 \xrightarrow{\tau}_{\{\}} W_1 \mid S' \mid W_2'$. Still, the two outcomes of the divergence are a critical pair that have no reason to reconverge $W_1' \mid S' \mid W_2 \to P'$ and $W_1 \mid S' \mid W_2' \to P'$ if W_i' are arbitrary behaviours. To preserve confluence we must block the environment from consuming the single prefix $w.S'$ twice. The problem is that the synchronisations $W_1 \mid S \xrightarrow{\tau}_{H_1} W_1' \mid S'$ and $S \mid W_2 \xrightarrow{\tau}_{H_2} S \mid W_2'$ have empty blocking

[4] In the pure λ-calculus the diamond property holds for β-reduction without any assumptions. We use the term "weak" confluence, because of the side-condition of non-interference.

sets $H_i = \{\,\}$ which does not prevent the second writer W_{3-i} to be added in parallel. If $w \in H_i$, then the rules (Par_{3-i}) would automatically prevent the second reader. Now, suppose we extend the action rule (Act) to expose the singleton nature of a prefix, like by (Act^+), $\alpha.P \xrightarrow{\alpha}_{\{\alpha\}} P$ we would not only prevent a second concurrent writer but already a single writer. The transitions $W_1 \xrightarrow{\overline{w}}_{\{\overline{w}\}} W_1'$ and $S \xrightarrow{w}_{\{w\}} S'$ would not be able to communicate, violating both side-condition $\{\overline{w}\} \cap \overline{\mathsf{iA}}(S) = \{\,\}$ and $\{w\} \cap \overline{\mathsf{iA}}(W_1) = \{\,\}$ of (Par_3). The issue is that in the setting of $\mathsf{CCS}^{\mathsf{CW}}$ (and $\mathsf{CCS}^{\mathsf{Ph}}$) the blocking set H of a transition $P \xrightarrow{\alpha}_H P'$ expresses a priority rather than an interference. An action cannot take priority over itself without blocking itself. Therefore, in $\mathsf{CCS}^{\mathsf{CW}}$ (and $\mathsf{CCS}^{\mathsf{Ph}}$) we always have $\alpha \notin H$. Here, we need to apply a weaker interpretation. The actions in H only prevent α from synchronising with a matching $\overline{\alpha}$ in the environment, if they are executed in a *different* (and thus competing) prefix from the one that executes $\overline{\alpha}$. Actions in H *interfere* with α in the sense that they are intial actions of P sharing the same control thread in which α is offered. Thus, we want to avoid a situation in which the environment synchronises with $\overline{\alpha}$ in the presence of *another distinct* communication on some $\beta \in H$. For then we would have a race for the behaviour of the thread shared by α and β in P.

To fix this, we need to refine the side-conditions $H_i \cap \overline{\mathsf{iA}}(P_{3-i})$ in (Par_3) to check not *all* initial actions $\overline{\mathsf{iA}}(P_{3-i})$ but only those initial actions of P_{3-i} that are *competing* with the matching transitions $P_1 \xrightarrow{\alpha_1}_{H_1} P_1'$ and $P_2 \xrightarrow{\alpha_2}_{H_2} P_2'$ with $\alpha_1 = \overline{\alpha}_2$, involved in the communication $P_1 \,|\, P_2 \xrightarrow{\tau}_{H_1 \cup H_2} P_1' \,|\, P_2'$ that we want to protect. We fix this by refining the information exposed by a transition. More specifically, we extend each transition $P \xrightarrow[K]{\alpha}_H P'$ by the set K of initial actions that are offered in competing prefixes of the same thread as α or by threads concurrent to the thread from which α is taken. Then, the set of all initial actions is simply $\mathsf{iA}(P) = K \cup \{\alpha\}$. Let us call K the *initial environment* of the transition. From the description, we expect the invariant that $H \subseteq K$.

▸ Remark 25. Note that we cannot simply define $K \stackrel{def}{=} \mathsf{iA}(P) - \{\alpha\}$, because α could be contained in K. For example, take $P \stackrel{def}{=} \alpha.0 \,|\, \alpha.0$, where $P \xrightarrow[\{\alpha\}]{\alpha}_{\{\alpha\}} \alpha.0$. The blocking set is $H = \{\alpha\}$ since the prefix is volatile and must be protected from a competing consumptions. Also, we need α to be in the initial environment $K = \{\alpha\}$, because the transition's action α is offered a second time in a concurrent thread. This is different from the set $\mathsf{iA}(P) - \{\alpha\} = \{\alpha\} - \{\alpha\} = \{\,\}$. This shows that we must compute the initial environment explicitly as an extra label with each transition.

Now we can reformulate the parallel rule as

$$\frac{P_i \xrightarrow[K_i]{\alpha_i}_{H_i} P_i' \qquad \alpha_1 = \overline{\alpha}_2 \qquad H_i \cap \overline{K}_{3-i} = \{\,\} \qquad \text{for } i \in \{1,2\}}{P_1 \,|\, P_2 \xrightarrow[K_1 \cup K_2]{\tau}_{H_1 \cup H_2} P_1' \,|\, P_2'} \,(Par_3^+)$$

and achieve the same as before but can accommodate the action rule (Act^+) that introduces a "reflexive precedence".

A.2.2 Fix 2

In the previous subsection we have seen how priorities can be beneficial to approaching to a deterministic scheduling in an intrinsic non-deterministic PA. The priority blocking eliminates the competition between mutually pre-empting actions and the remaining independent actions naturally commute. Now another question naturally arises: "does this mean that if we replace the non-deterministic choice $P + Q$ by a prioritised choice, denoted by $P \triangleright Q$, do

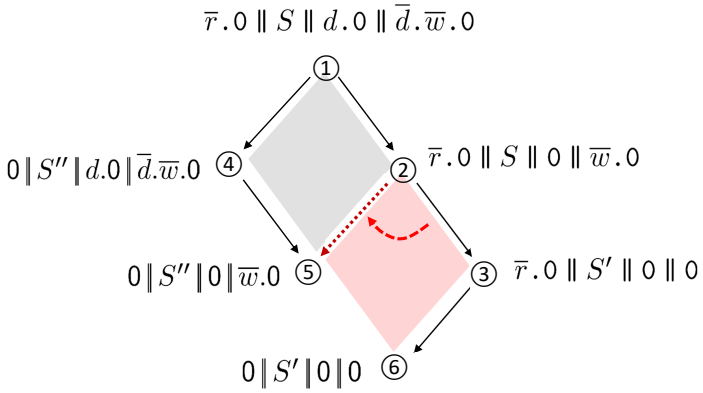


Figure 7 The store S in composition with a reader $\overline{r}.0$ and a delayed writer $d.0 \mid \overline{d}.\overline{w}.0$. The behaviour has two incongruent final outcomes, a store S'' with error in state ⑤ and an error-free store S in state ⑥. The transition system is not confluent. The reason is that transition labelled $\tau{:}\{w\}$ is admissible in state ① but not anymore in state ②. A solution is to block the reading already in state ①. In other words, the priority blocking (write-before-read) that kicks in at state ② needs to be detected already at state ①. Again, all transitons are silent with labels omitted.

we obtain a determinate process algebra?" Indeed, we can show that in the "asynchronous" fragment of CCS, i.e. where all prefixes $\alpha.P$ are of form $\alpha.0$, confluence can be granted: On the contrary, in the "synchronous" fragment of CCS, where actions sequentially guard new actions, we run into a problem of non-monotonicity. A communication between prefixes may trigger new actions that block another communication that was enabled before. In other words, it is not guaranteed that independent actions remain admissible. As an example, in the above reader-writer scenario, the actions a_i $(i \in \{1,2\})$ may become blocked in processes P_{3-i}. Now, consider a parallel composition $R \mid S^* \mid W^*$ in which a modified store $S^* \stackrel{def}{=} S \mid d.0$ offers another concurrent method action d on the side in parallel. Let the writer W^* be delayed by a synchronisation with this concurrent action d, i.e., $W^* \stackrel{def}{=} \overline{d}.W \stackrel{def}{=} \overline{d}.\overline{w}.0$. The scheduling sequences are depicted in Fig. 7. We can still derive $S \mid d.0 \xrightarrow[\{w,d\}]{r}_{\{r,w\}} S'' \mid d.0$ but also $S \mid d.0 \xrightarrow[\{r,w\}]{d}_{\{d\}} S$. The reading synchronisation $R \mid S \mid d.0 \xrightarrow[\{w,d\}]{\tau}_{\{\overline{r},r,w\}} S'' \mid d.0$ can now take place by (Par_3) before the write to give the following reduction sequence: $R \mid S \mid d.0 \mid W^* \xrightarrow[\{w,d,\overline{d}\}]{\tau}_{\{\overline{r},r,w\}} S'' \mid d.0 \mid W^* \xrightarrow[\{r,e\}]{\tau}_{\{d\}} S'' \mid W$. The first reduction is justified by rule (Par_1), because the side-condition $\{w\} \cap \overline{\mathsf{iA}}(W) = \{w\} \cap \{d\} = \{\,\}$ is satisfied. The blocking side-condition which only looks at the moment when the read is happening does not see the write action $\overline{w}$ by W^* hidden behind its initial action $\overline{d}$. It lets process R access and read the store S. As before, however, the following reduction sequence is allowed, too: $R \mid S \mid d.0 \mid W^* \xrightarrow[\{\overline{r},r,w\}]{\tau}_{\{d\}} R \mid S \mid W \xrightarrow[\{\overline{r},r\}]{\tau}_{\{w,\overline{w}\}} R \mid S' \xrightarrow[\{w\}]{\tau}_{\{\overline{r},r,w\}} S'$ in which the write action happens before the read action. This creates two diverging reduction sequences (incoherent schedules) in which the final store is S' or S'', non-deterministically.

Our insight is that the blocking side-conditions of the standard Camilleri-Winskell theory [7], described above, are too weak in scheduling control for our purposes and to guaranteeing a weak Church-Rosser property [18]. The problem is that in the side-condition of rule Par_2 we are only looking at the initial actions of Q. This is why the read action is blocked out of state ② in Fig. 7 but not out of state ①. In state ① there is no pending initial write. The write action $\overline{w}$ only becomes initial when the parallel processes $\overline{d}.0 \mid d.\overline{w}.0$ have interacted on action d and advanced together with a silent action τ, i.e. $\overline{d}.0 \mid d.\overline{w}.0 \xrightarrow{\tau} 0 \mid \overline{w}.0$.

In order to see that the read action r blocked by $\{w\}$ out of state ① should not be enabled, we must follow all sequences of interactions that can possibly happen concurrently to the read. The read is to be blocked, because after one silent τ-step, in the reachable configuration of state ②, the write action w in the blocking set $\{w\}$ of a silent action, meets a matching initial partner $\overline{w}$.

This suggests a strategy to fix up the standard theory. We accumulate, along with the *upper* blocking set H of a transition the following: a *lower* blocking set L and the context of all concurrent processes E that run in competition with the transition. From these we can find out what blocking actions might be generated after any number of silent actions. The initial environment K of a transition then is $H \cup L \cup \mathsf{iA}(E)$. This finally gives a transition relation of the form $P \xrightarrow[E]{a}_{H \cup L} P'$. The label E, called the *reduction context*, is accumulated in the parallel compositions of P. The rules (Act) and $(PriSum_i)$ are the one presented above. Further, let $\mathsf{iA}^\tau(E)$ be informally the set of all actions that can become initial along τ-sequences from E, which obviously includes $\mathsf{iA}(E)$ (formally defined later on). We will call the set $\mathsf{iA}^\tau(E)$ the set of *weak initial* actions of E. The rules (Par_1^*) and (Par_2^*) need to be reformulated. Rule (Par_1^*), propagating an action in a parallel composition, is as follows:

$$\frac{P \xrightarrow[E]{a}_{H \cup L} P' \qquad E' = E \,|\, Q \qquad H \cap \overline{\mathsf{iA}^\tau}(E') = \{\,\}}{P \,|\, Q \xrightarrow[E']{a}_{H \cup L} P' \,|\, Q} \; (Par_1^*)$$

Rule (Par_2^*) is symmetric to (Par_1^*). Rule (Par_3), for the synchronisation of an action α and its co-action $\overline{\alpha}$ using these weak initial sets, is reformulated as follows:

$$\frac{P \xrightarrow[E_1]{a}_{H_1 \cup L_1} P' \qquad Q \xrightarrow[E_2]{\overline{\alpha}}_{H_2 \cup L_2} Q' \qquad E = E_1 \,|\, E_2 \qquad \forall i.\, H_i \cap (\overline{\mathsf{iA}^\tau}(E) \cup L_{3-i}) = \{\,\}}{P \,|\, Q \xrightarrow[E]{\tau}_{H_1 \cup H_2 \cup L_1 \cup L_2} P' \,|\, Q'} \; (Par_3^*)$$

We are now ready to revisit our running example. By (Par_2^*), since

$$R \xrightarrow[0]{\overline{r}}_{\{r\} \cup \{\,\}} R' \text{ and } S \,|\, d.0 \xrightarrow[d.0]{r}_{\{w,r\} \cup \{\,\}} S'', \text{ we get } R \,|\, S \,|\, d.0 \xrightarrow[d.0]{\tau}_{\{w,r\} \cup \{\,\}} S'' \,|\, d.0.$$

The side-condition of (Par_3^*) is satisfied since the "weak" initial action set (where τ actions are dropped) $\mathsf{iA}^\tau(d.0) = \{d\}$. Thus, both $\{r\} \cap \overline{\mathsf{iA}}^*(d.0) = \{\,\}$ as well as $\{w, r\} \cap \overline{\mathsf{iA}}^*(d.0) = \{\,\}$. If now want to shunt this past the writer process W^* using rule (Par_1^*), then we get stuck. Since $d.0 \,|\, W^* \xrightarrow[0]{\tau}_{\{d,\overline{d}\} \cup \{\,\}} W$ and $\overline{w} \in \mathsf{iA}(W)$, we find $\overline{w} \in \mathsf{iA}^\tau(d.0 \,|\, W^*)$ and thus $\{w\} \cap \overline{\mathsf{iA}}^*(d.0 \,|\, W^*) \neq \{\,\}$. Therefore, the side-condition of (Par_1^*) is violated and the reading interaction of $R \,|\, S \,|\, d.0$ cannot be composed with W^*. This is what we want. The transition from state ② to state ③ in Fig. 7 is now blocked. On the other hand, the reduction $R \,|\, S \,|\, d.0 \xrightarrow[R \,|\, S]{d}_{\{d\} \cup \{\,\}} R \,|\, S$ can be composed with $W^* \xrightarrow[0]{\overline{d}}_{\{\overline{d}\} \cup \{\,\}} W$ via (Par_3^*) to make a reduction $R \,|\, S \,|\, d.0 \,|\, W^* \xrightarrow[R \,|\, S]{\tau}_{\{d,\overline{d}\} \cup \{\,\}} R \,|\, S \,|\, W$. Regarding the side-condition, we notice that the weak initial action set $\mathsf{iA}^\tau(R \,|\, S)$ of the reduction context is irrelevant, because it does not contain d or $\overline{d}$. In configuration $R \,|\, S \,|\, W$ (state ② in Fig. 7), the reader R cannot interact with S. The associated reduction $R \,|\, S \,|\, W \xrightarrow[W]{\tau}_{\{r,w\} \cup \{\,\}} S'' \,|\, W$ is blocked by the reduction environment, because $w \in \overline{\mathsf{iA}}^*(\overline{w}.W')$. For our further discussion, let us call the proposed change in the side-conditions leading to (Par_i^*) from (Par_i^*) a process algebra with *strong priorities*[5] and let's call the traditional process algebra with priorities, introduced by

[5] The terminology is suggested by the fact that strong priorities use weak initial sets as rule premises while weak priorities are based on strong initial sets. Rules are implications, so weakening the antecedent makes a rule logically stronger.

[7, 30], as process algebra with *weak priorities*.

A.2.3 Digression: Strong Priorities lead to some Impredicativity Issues

On the face of it, strong priorities may seem rather a natural modification to make. However, they amount to a significant change of the game compared to the standard weak priorities presented in the literature. For they break the standard method of defining the transition relation inductively as the least relation closed under a finite set of production rules. The key problem is the logical cycle in the structural definition of the operational semantics: the rules to generate the transitions ensure $H \cap \overline{\mathsf{iA}}^*(E) = \{\}$, i.e., that none of the blocking actions H matches with the transitively reachable initial actions of the reduction context E which itself is extracted as a part of P.

Obviously, to compute $\mathsf{iA}^\tau(E)$ we need to know the full transition relation for E. In other words, the existence of a transition of a process P depends on the absence of certain transitions of a certain part of process P. If P is defined recursively or via repetition then the behaviour of a part of P might depend on the behaviour of P itself. Does such a self-referential semantical definition make sense at all? How do we ensure the absence of a Russell-paradox style relation, by defining a process $\mathsf{rec}\, p.\, P$ to have a transition to P' if it does not have a transition to P' for a suitable P'? Such would be plainly inconsistent at the meta-level, because it would not only define simply an empty transition relation[6], but it would not define a unique semantic relation at all.

In the standard theory of weakly prioritised process algebra as described in the cited references of CCS with priorities, the problem does not exist, because the set of (strong) initial actions $\mathsf{iA}(E)$ can be determined by structural recursion on E without any reference to the transition relation. Unfortunately, for strong prioritized process algebra as defined in this paper, this is not possible anymore. So, how do we make sense of transition-generating rules (Par_i^*) that depend on negative premises? One option is to treat the negation in the side-condition $H \cap \overline{\mathsf{iA}}^*(E) = \{\}$ constructively rather than classically, say like negation in normal logic programming. This would amount to building a second independent (inductive) rule system to generate constructive judgements about the absence of weakly reachable transitions. Trouble is that there is no canonical semantics of constructive negation, even in logic programming[7].

Some progress has been made for unifying the proposals in a game-theoretic and intuitionistic setting which works well for special situations like the constructive semantics of Esterel [3] or StateCharts [14]. However, playing with constructive negation at the meta-level will give a variety of semantics whose adequacy might be difficult to judge in practical application contexts.

A more perspicuous attack is to keep the classical set-theoretic interpretation but use an over-approximation $L \cap \overline{\mathsf{fiA}}^*(E) = \{\}$ where $\mathsf{fiA}^*(E) \supseteq \mathsf{iA}^\tau(E)$ is intuitively a superset of weakly reachable initial sender actions obtained from a relaxed transition relation, formally defined later in this paper. A natural over-approximation that can be defined independently is the "free speculative" scheduling that ignores all priorities. The stronger side-condition $L \cap \overline{\mathsf{fiA}}^*(E) = \{\}$ forces the absence of a blocking sender action under the worst-case assumption of priority-free scheduling. Obviously, this is sound but will have the effect that some programs block that could make progress under a tighter approximation of the sets

[6] As an example consider the unguarded process $\mathsf{rec}\, p.p$ in the standard inductive theory. Its behaviour is well-defined as the empty transition relation.

[7] Different well-known semantics are negation-by-failure, stable semantics, supported model semantics.

$\mathsf{iA}^\tau(E)$. Between $\mathsf{fiA}^*(E)$ and $\mathsf{iA}^\tau(E)$ there be a whole range of different semantics that may or may not be equivalent to specific constructive semantics of negation. Exploring the range of options systematically will certainly be a worthwhile research exercise likely to cover much new ground in the theory of process algebras with priorities.

A.3 Synchronous CCS with Strong Priories and Clocks is Confluent

We argue that the use of priorities for determinacy calls for a combination of strong priorities with a second important concept that has been investigated in the literature before, but independently. This is the well-known notion of a *clock*, from Timed Process Algebras, like e.g. TPL [15] and PMC [12]. Intuitively, a clock σ is a "broadcast action" in the spirit of Hoare's CSP [16] that acts as a synchronisation barrier for a set of concurrent processes. It bundles each processes' actions into a sequence of *macro-steps* that are aligned with each other in a lock-step fashion. During each macro-step, a process executes only a temporal slice of its total behaviour, at the end of which it waits for all other processes in the same *clock scope* to reach the end of the current phase. When all processes have reached the barrier, they are released into the next round of computation.

This scheduling principle is well-known from computer hardware like sequential circuits [33] and VHDL [17], or synchronous languages like Esterel [3], Lustre [13], SCCharts [36], BSP [34], just to mention a few. For our purposes, clocks are the universal trick to break causality cycles in priority scheduling, when processes need to communicate with each other in iterated cycles (so-called *ticks* or *macro-steps*) while maintaining determinacy.

To get this off the ground we introduce a clock broadcast action σ to separate each round of communication. We define the recursive processes $W \stackrel{def}{=} d.0 \,|\, \overline{d}.\overline{w}.\sigma.W$ and $R \stackrel{def}{=} \overline{r}.\sigma.R$ for one write and one read per macro-step. Assuming the content of the memory variable is reset for every tick, we can define S as follows $S \stackrel{def}{=} w.S' + r{:}\{w\}.S'' + \sigma{:}\{w,r\}.S$ with $S' \stackrel{def}{=} r.S'' + \sigma{:}\{r\}.S'$ and $S'' \stackrel{def}{=} w.\mathrm{fail} + r{:}\{w\}.S'' + \sigma{:}\{w,r\}.S$. The clock always takes *lowest* priority among all other rendez-vous actions out of a state. The idea is that the clock should fire only if the system has stabilised and no other admissible data actions are possible. This is called *maximal progress* and it is a standard assumption in timed process algebras [15, 8]. The fairly standard rules for clock actions, inspired by timed process algebras are then:

$$\frac{}{\sigma.P \xrightarrow[0]{\sigma}_{\{\}} P}\,(Clk_1) \qquad \frac{P \xrightarrow[R_1]{\sigma}_{L_1} P' \qquad P \xrightarrow[R_2]{\sigma}_{L_2} P' \qquad (L_1 \cup L_2) \cap \mathsf{iA}^\tau(R_1 \,|\, R_2) = \{\,\}}{P \,|\, Q \xrightarrow[R_1 \,|\, R_2]{\sigma}_{L_1 \cup L_2} P' \,|\, Q'}\,(Clk_2)$$

Those two rules enforces global synchronisation and make the clock deterministic. As such, the evaluation of $R \,|\, S \,|\, W$ "prunes" all reductions that could lead to non confluent reductions. Fig. 8 shows how higher priorities can introduce determinism in scheduling. The importance of the clock σ is that its associated synchronisation rule (Clk_2) enforces, in a quite natural way, the barrier explicitly in the SOS rather than making it an artefact of a specific synchronisation process, as such obliging the programmer to an extra effort. The fact that the broadcasting clock is hard-wired into the rules can now be exploited for the computation of the *weak initial action sets*, defined formally $\mathsf{iA}^\tau(R)$. More precisely, we only need to predict the behaviour of R for the current macro-step, i.e., up to the next synchronization of σ by R. For a transition $P \xrightarrow[R]{a}_L P'$ we know that all actions taking place after σ in R are definitely observed to happen in the next macro step and thus not in competition with the current action a under blocking set L. In this fashion, the clock acts as as a *stopper sentinel* for prediction and evaluation of the blocking set L relative to the reduction environment R. If every recursion is instantaneous, then the search space becomes finite.

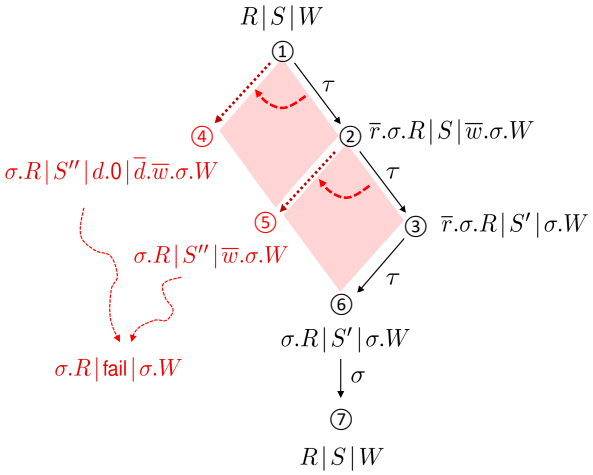


Figure 8 Strong Priorities introduce confluence in scheduling.

As a final remark, for the purposes of this paper, adding synchrony and determinism in process algebras, we are not (and indeed we do not need to be) interested in higher-order calculi, like e.g. the π-calculus [24, 25], where processes are *first-class* objects, i.e. sent and received as a values, as in $a!P.Q$ and $a?x.(x\,|\,Q)$. In the next sections, we present the syntax, the structural operational semantics à la Plotkin, we set up all the definitions and the metatheory necessary to formally prove the claimed informal properties.

B Examples, new and reloaded

This section contains a number of reloaded and new examples in view of the definitions and results presented in the previous section.

The weakly-enabled transitions of unclocked, free processes coincide with the semantics of genuine Milner's CCS [22], as stated in the following easy exercise.

▸ **Exercise 26** (On Def. 3). *If P is unclocked and free, then a transition $P \xrightarrow[R]{\alpha}_H Q$ is weakly enabled iff $P \xrightarrow{\alpha} Q$ is admissible, i.e., iff it is derivable in the original Milner's SOS for CCS.*

In fact, one can show that the semantics of weak enabling generates the theory of $\mathsf{CCS}^{\mathsf{Ph}}$ [27]. To be more precise, let us call a process P *irreflexive* if no action prefix $\alpha{:}H.Q$ occurring in P has $\alpha \in H$, i.e., blocks itself. In the semantics of $\mathsf{CCS}^{\mathsf{Ph}}$, such *self-blocking* prefixes do not contribute any transitions. Thus, $\mathsf{CCS}^{\mathsf{Ph}}$ captures a theory of irreflexive processes. We leave as a less easy exercise that all weakly enabled transitions between irreflexive processes are precisely the ones allowed in $\mathsf{CCS}^{\mathsf{Ph}}$.

▸ **Exercise 27** (On Def. 3). *If P is unclocked and irreflexive, then a transition $P \xrightarrow[R]{\alpha}_H Q$ is weakly enabled iff $P \xrightarrow{\alpha}_H P'$ is derivable in the operational semantics of $\mathsf{CCS}^{\mathsf{Ph}}$.*

Thus, from the point of view of $\mathsf{CCS}^{\mathsf{spt}}$, theories of CCS and $\mathsf{CCS}^{\mathsf{Ph}}$ correspond to theories of free and irreflexive processes, respectively, under weak enabling. Here we suggest two extension to this classical work. Firstly, we apply a stronger scheduling restriction, called *strong enabling* to achieve confluence and secondly we make use of self-loops to control sharing.

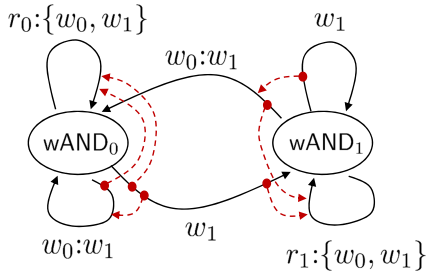
▸ **Example 28** (On Def. 16). If $A \not\equiv B$ then $a.A + a.B$ is not coherent while $a{:}a.A \,|\, a{:}a.B$ is coherent. Conflicting choices on the same label in a single thread cannot be externally observed and thus cannot be deterministically scheduled via their externally observable properties. In coherent processes such choices are assumed to be resolved internally by each thread itself, rather than through the precedence relation. Note that the process $a{:}a.A + a{:}a.B$ in which the two action prefixes are self-blocking, is not coherent. The reason is that the two identical (self-blocking) transitions lead to structurally distinct states $A \not\equiv B$. Thus, the confluence condition expressed in Def. 16 applies and A and B should be reconvergent with transitions $A \xrightarrow{a} Q'$ and $B \xrightarrow{a} Q'$. But this is not guaranteed for arbitrary A and B. On the other hand, a single self-blocking prefix $a{:}a.A$ is coherent. Without the self-blocking, $a.A$ is only coherent if process A permits infinite repetition of a as in $A \stackrel{def}{=} a.A$ or $A \stackrel{def}{=} a \,|\, A$.

The precedence relation cannot resolve the choice of a prefix with itself. The precedence relation can resolve the choice between *distinct* actions prefixes. For instance, if $a \neq b$ then $a.A + b.B$ is not coherent, for any A and B. However, both $a{:}a.A \,|\, b{:}b.B$ and $a{:}a.A + b{:}\{a,b\}.B$ are coherent. In the former case, the actions are concurrently independent and so confluence occurs naturally. In the latter case, the choice is resolved by precedence: No reconvergence is required by coherence Def. 16 as the actions are interfering with each other. ◂

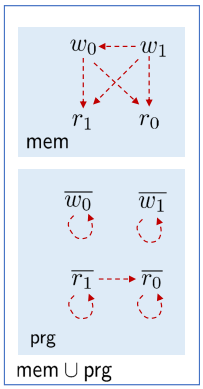
▸ **Example 29** (On coherence, pivotable and policies). The process $P_1 \stackrel{def}{=} a \,|\, \overline{b} \,|\, \overline{a}{:}b + b$ is coherent and pivotable with a policy π such that $\pi \Vdash b \dashrightarrow \overline{a}$ and $\pi \Vdash \overline{b} \diamond a$. The rendez-vous synchronisation $\tau = a \,|\, \overline{a}$ is blocked by the presence of the offering $\overline{b}$ from a *third* thread. We have $P_1 \xrightarrow[\overline{b}]{\tau}_{\{b\}} \overline{b}$, which is not weakly enabled, because $\{b\} \cap \overline{\mathsf{iA}}(\overline{b}) \neq \{\,\}$. The blocking is due to the reaction environment $\overline{b}$ of the transition. For contrast, the process $P_2 \stackrel{def}{=} (a + \overline{b}) \,|\, (\overline{a}{:}b + b)$ is pivotable but not coherent. In our semantics, the synchronisation of a and $\overline{a}{:}b$ in P_2 is blocked by the side-condition of rule (Par_3) which introduces τ into the blocking set $P_2 \xrightarrow[0]{\tau}_{\{b,\tau\}} 0$ because $\{b\} \cap \overline{\mathsf{iA}}(a + \overline{b}) \not\subseteq \{\overline{a}\}$. This transition is not weakly enabled, because of $\{b,\tau\} \cap (\mathsf{iA}(0) \cup \{\tau\}) \neq \{\,\}$. We can make the pivotable P_1 also coherent by forcing a sequential ordering between a and $\overline{b}$ in the first thread, say $P_3 \stackrel{def}{=} a.\overline{b} \,|\, (\overline{a}{:}b + b)$, then the synchronisation is even strongly enabled $P_3 \xrightarrow[0]{\tau}_{\{b\}} \overline{b}$. If we introduce a precedence into the first thread, say $P_4 \stackrel{def}{=} (a{:}b + \overline{b}) \,|\, (\overline{a}{:}b + b)$, then we are also coherent but no longer pivotable. Like for P_2 the synchronisation $\tau = a \,|\, \overline{a}$ is blocked and not weakly enabled, because of the introduction of τ into the blocking set by the side condition of Par_3. ◂

▸ **Example 30** (On Normal form). For the process $P_1 \stackrel{def}{=} (a.c \,|\, b.d \,|\, \overline{a} \,|\, \overline{b}) \setminus \{a,b\}$ we have $(a.c \,|\, d \,|\, \overline{a}) \setminus \{a,b\} \xleftarrow{\tau} P_1 \xrightarrow{\tau} (c \,|\, b.d \,|\, \overline{b}) \setminus \{a,b\}$ where the continuation processes $(c \,|\, b.d \,|\, \overline{b}) \setminus \{a,b\}$ and $(a.c \,|\, d \,|\, \overline{a}) \setminus \{a,b\}$ are not structurally equivalent. They are not even behaviourally equivalent. However, both reductions are concurrently independent and commute with each other. We can execute both rendez-vous synchronisations $\tau = a \,|\, \overline{a}$ and $\tau = b \,|\, \overline{b}$ in any order, the process P converges to a unique final outcome $(c \,|\, d) \setminus \{a,b\}$. Similarly, in the multi-cast process $P_2 \stackrel{def}{=} (a.c \,|\, a.d \,|\, \overline{a} \,|\, \overline{a}) \setminus \{a\}$ the reduction behaviour is not structurally deterministic but nevertheless generates a unique normal form. ◂

▸ **Example 31** (On Clock-hiding). The process $P \stackrel{def}{=} \sigma + a{:}\sigma \,|\, \sigma$ offers a choice of two transitions and $P \xrightarrow[\sigma]{a}_{\{\sigma\}} \sigma$. These transitions interfere with each other as the σ-transition blocks the a-transition. We expect the a-transition to be blocked. This must happen when the clock is hidden and thus generates a silent step. Indeed, when we hide the clock, we change the blocking σ into a blocking $\sigma \,/\, \sigma = \tau$ and obtain $P \,/\, \sigma \xrightarrow[\sigma \,/\, \sigma]{a}_{\{\tau\}} \sigma \,/\, \sigma$. Now the τ in the blocking set eliminates the a-transition under enabling semantics, in particular under constructive



(a) A Boolean Read-Write Memory with coherent states $\mathsf{wAND}_0 : \mathsf{mem}$ and $\mathsf{wAND}_1 : \mathsf{mem}$.



(b) The mem ∪ prg Pivot Policy.

Figure 9 A boolean memory process.

scheduling. So far so good. However, we can replace the blocking set $\{\sigma\}$ by $\{\tau\}$ only if the clock is actually executable in the concurrent environment $0 \,|\, \sigma$. But it is not. In the process $Q \stackrel{def}{=} (\sigma + a{:}\sigma \,|\, b.\sigma) \,/\, \sigma$ the local clock σ also blocks the action a by priority. For the clock to become enabled, however, it needs the synchronisation with an external action $\overline{b}$. The naive hiding rule would generate the transition $Q \xrightarrow[(b.\sigma)\,/\,\sigma]{a}_{\{\sigma\}} (b.\sigma) \,/\, \sigma$. Under constructive scheduling we would find $\sigma \in \overline{\mathsf{iA}}^*(b.\sigma)$. This could be taken as justification to introduce τ into the blocking set. It provides enough information for us to be able to see that the parallel composition $Q \,|\, \overline{b}$ will not permit the a-transition any more.

▸ **Example 32** (On Pivotable Processes)**.** The process $a + b{:}c \,|\, \overline{a} + \overline{b}{:}\overline{a}$ is pivotable. If P conforms to π then the only precedences forced are $\pi \Vdash c \dashrightarrow b$ and $\pi \Vdash \overline{a} \dashrightarrow \overline{b}$ while it satisfies $\pi \Vdash \overline{c} \diamond \overline{b}$ and $\pi \Vdash a \diamond b$. However, the process is not input-scheduled, because it has $\pi \Vdash \overline{a} \dashrightarrow \overline{b}$ while $\pi_{is} \Vdash \overline{a} \diamond \overline{b}$, assuming $a \neq b$.

▸ **Example 33** (On Coherent and Pivotable Processes)**.** The process $P_1 \stackrel{def}{=} \sigma + a{:}\sigma \,|\, \overline{a}$ is coherent and pivotable, with policy $\pi \Vdash \sigma \dashrightarrow a$ and $\pi \Vdash \sigma \diamond \overline{a}$. It has both $a, \overline{a} \in \mathsf{iA}(P)$ but $\sigma \notin \mathsf{iA}(P)$ since the clock is not offered by the parallel process $\overline{a}$. The synchronisation $\tau = a \,|\, \overline{a}$ can go ahead. On the other hand, the process $P_2 \stackrel{def}{=} \sigma + a{:}\sigma \,|\, \overline{a} + \sigma{:}\overline{a}$ has all three actions $\sigma, a, \overline{a} \in \mathsf{iA}(P_2)$ initial. It is coherent but not pivotable. Since both a and $\overline{a}$ are in a premption relationship with σ, P_2 does not satisfy concurrent independence of σ with neither a nor $\overline{a}$. Formally, any policy for P_2 must have $\pi \Vdash \sigma \dashrightarrow a$ and $\pi \nVdash \overline{a} \dashrightarrow \sigma$ violating the pivot requirement. The same is true of the process $P_3 \stackrel{def}{=} \sigma + (a{:}\sigma \,|\, \overline{a})$ which can perform both σ and a (silent) reduction that has σ in its blocking set.

▸ **Example 34** (On Read/Write Process, reloaded)**.** Figure 9 show that the processes wAND_v conform to a policy mem with $\mathsf{mem} \Vdash w_i \dashrightarrow r_j$ for $i \neq j$ and $\mathsf{mem} \Vdash w_1 \dashrightarrow w_0$. Obviously, mem is a pivot policy. The processes wAND_v are also coherent. One verifies by simple case analysis that for any derivative Q of wAND_v, if $Q \xrightarrow[0]{\ell}_H Q'$ and $\ell \notin H$ then there is a repetition $Q' \xrightarrow[0]{\ell}_{H'} Q''$ with $H' = H$. This yields Def. 16(1). Secondly, suppose $Q \xrightarrow[0]{\ell_1}_{H_1}$ and $Q \xrightarrow[0]{\ell_2}_{H_2} Q_2$. Then, obviously, if $\ell_1 \neq \ell_2$ the two transitions are never interference-free, because of the choice of blocking sets which orders all summation prefixes in a linear precedence order. If $Q_1 \not\equiv Q_2$ the necessarily $\ell_1 \neq \ell_2$. Thus, Def. 16(2) is satisfied, too, for trivial reasons. So, the memory $\mathsf{wAND}_v : \mathsf{mem}$ is pivotable. Note that wAND_v is not a discrete process, since none of

its actions is self-blocking. We can put the memory in parallel with arbitrarily many writers and readers. Consider a parallel composition $\mathsf{wAND}_v \mid P$ where P is any single-threaded process that synchronises via $\overline{w_i}$ for writing and $\overline{r_i}$ for reading. Each writing action of P is a self-blocking prefix $\overline{w_i}{:}\overline{w_i}.Q$ and each reading is a choice $\overline{r_1}{:}\overline{r_1}.R_1 + \overline{r_0}{:}\{\overline{r_0}, \overline{r_1}\}.R_0$ where the continuation processes R_0 and R_1 are selected according to the content of the memory, by matching with the r_0 action of wAND_0 or the r_1 action of wAND_1. If the memory would offer both a synchronisation on r_0 and r_1 (which it does not), the reader would take the match on r_1, because of the precedences. If Q, R_0 and R_1 follow the same principle, such a sequential program is a discrete process $P : \mathsf{prg}$ that is coherent and conformant to the policy prg with $\mathsf{prg} \Vdash \overline{r_1} \dashrightarrow \overline{r_i}$ and $\mathsf{prg} \Vdash \overline{r_0} \dashrightarrow \overline{r_0}$. Now we observe that the union $\mathsf{mem} \cup \mathsf{prg}$ of policies is a pivot policy. Since $\mathsf{wAND}_v : \mathsf{mem}$ and $P : \mathsf{prg}$ we have $\mathsf{wAND}_v : \mathsf{mem} \cup \mathsf{prg}$ as well as $P : \mathsf{mem} \cup \mathsf{prg}$. Therefore, by Prop. 51 we get $\mathsf{wAND}_v \mid P : \mathsf{mem} \cup \mathsf{prg}$ i.e., the composition is coherent and pivotable. In fact we can put the memory in parallel with arbitrarily many sequential programs coherent for prg and the composition $\mathsf{wAND}_v \mid P_1 \mid P_2 \mid \cdots \mid P_n$ will be coherent for $\mathsf{mem} \cup \mathsf{prg}$. In particular, all reductions will be confluent under strong enabling. Technically, what happens is that first all writers 1, synchronising as $w_1 \mid \overline{w_1}$, will go ahead. Then all writers of 0, synchronising as $w_0 \mid \overline{w_0}$ become enabled. Only then the readers with be able to retrieve the final value via $r_0 \mid \overline{r_0}$ or $r_1 \mid \overline{r_1}$ rendez-vous synchronisations.

Observe that the pivot policy $\mathsf{mem} \cup \mathsf{prg}$ permits a choice (and precedence) between both the receiver actions w_0, w_1 but also between the sender actions $\overline{r_0}$, $\overline{r_1}$. As a result, the combined memory and program, which is coherent for $\mathsf{mem} \cup \mathsf{prg}$, is not longer an input-scheduled process. This breaks the assumption on which the theory of classical prioritised process algebras like [7] or [30] are based. In reference to the former, Phillips writes:

> *They [Camilleri and Winskel] decide to sidestep this question by breaking the symmetry in CCS between inputs and outputs, and only allowing prioritised choice on input actions. We feel that this complicates the syntax and operational semantics, and should not be necessary. [27]*

Our example and analysis indeed highlights the usefulness of the more general setting suggested by Phillips. In order to explain the theory of deterministic shared memory we need a larger class of processes than those considered in $\mathsf{CCS}^{\mathsf{CW}}$.

The memory cell schedules readers after the writers and among the writers prefers to engage first with with those that set the value to 0, in the prioritised sum. This implements an Esterel-style resolution function for multi-writer convergecast, multi-reader broadcast of boolean signals. All concurrent readers receive the same value, which is the combined-AND of all values written by any concurrent writer. If any process writes a 1 it will win the competition. This is also known as a "wired-AND". Likewise, we can implement a memory cell that models a "wired-OR" if we swap the roles of w_0 and w_1 in the prioritised choices of wAND_v.

Note that the wired-AND behaviour only applies to concurrent threads. A single thread can use the wAND cell as a local memory and destructively update it sequentially as it wishes. Consider the difference: Both *concurrent* writers $\overline{w_1}{:}\overline{w_1} \mid \overline{w_0}{:}\overline{w_0} \mid \mathsf{wAND}_0 \mid R$ and $\overline{w_0}{:}\overline{w_0} \mid \overline{w_1}{:}\overline{w_1} \mid \mathsf{wAND}_0 \mid R$ will necessarily make the reader $R \stackrel{def}{=} r_0{:}r_1.A + r_1{:}\{r_0, r_1\}.B$ detect a value r_0 and therefore end in state A (wired-AND of w_0 and w_1). The two single-threaded *sequential* writings $\overline{w_1}{:}\overline{w_1}.\overline{w_0}{:}\overline{w_0} \mid \mathsf{wAND}_0 \mid R$ and $\overline{w_0}{:}\overline{w_0}.\overline{w_1}{:}\overline{w_1} \mid \mathsf{wAND}_0 \mid R$ will generate two different readings by R: The first is the reading r_0, leading to A. The second is the reading r_1, leading to B.

▸ **Example 35** (On ABRO, reloaded)**.** One can verify that the process ABRO is coherent and

pivotable.

- The choices in each of the threads R, A, B, O are fully resolved by the blocking sets, i.e., every pair of actions in the summations are distinct and in precedence relationships. Every rendez-vous prefix is self-blocking unless the action can be repeated with the same blocking in the continuation process. This happens for channels s and k in the recursions $\mathsf{O} \stackrel{def}{=} \ldots + s{:}k.\mathsf{O} + \ldots$ and $\mathsf{R}' \stackrel{def}{=} \overline{k}.\mathsf{R}' + \ldots$. Thus, the threads R, A, B, O themselves are coherent.
- The associated policy π, for which the threads R, A, B, O are coherent, enforces the precedences
  - $\pi \Vdash k \dashrightarrow \ell$ for $\ell \in \{k, a, b, s, t, \sigma\}$
  - $\pi \Vdash \ell \dashrightarrow \ell$ for $\ell \in \{r, k, a, b, \overline{s}, t, \overline{o}\}$
  - $\pi \Vdash s \dashrightarrow t$
  - $\pi \Vdash \ell \dashrightarrow \sigma$ for $\ell \in \{a, b, s, t\}$.

  The dual matching pairs for these precedences, however, are all concurrently independent. More precisely, we look at distinct labels, $\pi \Vdash \overline{k} \diamond \overline{\ell}$ for all $\ell \in \{a, b, s, t, \sigma\}$, $\pi \Vdash \overline{s} \diamond \overline{t}$ and $\pi \Vdash \overline{\ell} \diamond \sigma$ for $\ell \in \{a, b, s, t\}$.
- Thus the policy π (and thus $\pi \setminus \mathcal{R}$) is a pivot policy and so the parallel composition $\mathsf{A} \,|\, \mathsf{B} \,|\, \mathsf{O}$ is coherent for π. Further, π is precedence-closed for $\{s, t\}$ which means that ABO and also ABRO are coherent for π.

▸ **Example 36** (On Sequentiality à la Milner in CCS$^{\mathsf{spt}}$)**.** Our calculus CCS$^{\mathsf{spt}}$ does not have a generic operator for sequential composition, like CCS and unlike Esterel. This is not necessary, because sequential composition can be coded in CCS by action prefixing and parallel composition. More precisely, a sequential composition $P; Q$ is obtained as a parallel composition $([P]_t \,|\, t{:}t.Q) \setminus \{t\}$ where $[P]_t$ is the process P, modified so when it terminates, then the termination signal $\overline{t}$ is sent. The signal t will then trigger the sequentially downstream process Q. The operation $[P]_t$ can be implemented by a simple (linear) syntactic transformation of P. In the same way we can detect termination of a parallel composition $P = P_1 \,|\, P_2$. Since a parallel terminates if both threads terminate, we can define $[P]_t$ as follows $[P_1 \,|\, P_2]_t \stackrel{def}{=} ([P_1]_{s_1} \,|\, [P_2]_{s_2} \,|\, s_1{:}s_1.s_2{:}s_2.\overline{t}) \setminus \{s_1, s_2\}$ where s_1 and s_2 are fresh "local" termination signals. The "termination detector" $T_{seq} \stackrel{def}{=} s_1{:}s_1.s_2{:}s_2.\overline{t}$ first waits for P_1 to terminate and then for P_2, before it sends the signal $\overline{t}$ indicating the termination of the parallel composition. We could equally first wait for P_2 and then for P_1, if the difference is not observable from the outside. In cases where the termination detector needs to permit both processes P_1 and P_2 to terminate in any order, however, we need a different solution. We need to "wait concurrently" for s_1 or s_1, not sequentially as in T_{seq}. The obvious idea would be to use a *confluent sum* $T_{csum} \stackrel{def}{=} s_1{:}s_1.s_2{:}s_2.\overline{t} + s_2{:}s_2.s_1{:}s_1.\overline{t}$ as discussed in Milner [22] (Chap. 11.4). This does not work, because the sum must explicitly unfold all possible orders in which the termination signals can arrive and thus again creates an exponential descriptive complexity. Interestingly, in CCS$^{\mathsf{spt}}$ we can exploit precedence scheduling to obtain a robust and compositional solution. Consider the process $T_{par} \stackrel{def}{=} s.T_{par} + \overline{t}{:}\{s, \overline{t}\}$. It engages in an unbounded number of receptions on channel s without changing its state. At the same time, it offers to send the termination signal $\overline{t}$, but only if the action s is not possible. Under the weak enabling strategy, the composition $\overline{s} \,|\, \overline{s} \,|\, T$ will consume the two senders $\overline{s}$ and only then engage in t. In this way, we can use T_{par} to detect the *absence* of s which corresponds to the termination of the two environment processes $\overline{s} \,|\, \overline{s}$, in any order. The signal s could be the shared termination signal of each process P_1 and P_2 that is sent and consumed by T_{par} if and when the process terminates. If we execute the parallel composition $P_1 \,|\, P_2 \,|\, T_{par}$ under constructive enabling, the termination

signal $\overline{t}$ will not be sent until such time that both P_1 and P_2 have been able to send s to T. From the scheduling point of view, the composition $P_1 \,|\, P_2$ acts like a counter that is initialised to 2, representing the number of potential offerings of s initially in $P_1 \,|\, P_2$. It gets decremented each time one of P_1 or P_2 terminates. For instance, $\mathsf{0} \,|\, P_2$ and $P_1 \,|\, \mathsf{0}$ have one potential offering left, while $\mathsf{0} \,|\, \mathsf{0}$ has no more potential to send s. At this moment, the counter has reached 0 and $\mathsf{0} \,|\, \mathsf{0} \,|\, T_{par}$ can take the $\overline{t}$ action and thereby send termination. In sum, there are two important observations here: First, we find that precedences with constructive enabling permits us to code sequential composition of Esterel in a compositional fashion. Second, the coding of Milner's confluent sum T_{csum} is possible in $\mathsf{CCS}^{\mathsf{spt}}$ with linear descriptive complexity.

▸ **Example 37** (On Multi-Cast and Constructive Enabling)**.** Suppose $S_0 = a{:}e.S_0 + e.S_1 + \sigma{:}\{a, e\}.S_0$ is an Esterel Signal (Sec. 11) in its absent state, with abbreviations $e = \mathit{emit}$, $a = \mathit{abs}$ and $p = \mathit{pres}$. Its initial action $S_0 \xrightarrow[\mathsf{0}]{a}_{\{e\}} S_0$ to communicate absence does not block itself to permit multi-reader scenarios. A typical reader is of form $R = \overline{p}{:}\overline{p}.R_1 + \overline{a}{:}\{\overline{a}, \overline{p}\}.R_0$. This is a choice: if the signal is present we execute R_1, otherwise if it is absent we execute R_0. The two sender actions $\overline{p}$ and $\overline{a}$ are self-blocking, because these prefixes can only be consumed once. In essence, what we want is that there can be many senders of the value labels (readers) but at most one receiver (signal).

Consider a reader $C = \overline{a}{:}\overline{a}.\overline{e}{:}\overline{e}.C_1$ that wants to emit the signal when it finds it absent. The composition $S_0 \,|\, C$ is deterministic under strong enabling. $S_0 \,|\, C \xrightarrow[\mathsf{0}\,|\,\mathsf{0}]{\tau}_{\{e,\overline{a}\}} S_0 \,|\, \overline{e}{:}\overline{e}.C_1$. The blocking set contains two labels:

(1) First, $e \in \{e, \overline{a}\}$ indicates priority, i.e., that we do not want another concurrent thread sending $\overline{e}$, because the synchronisation for absence is based on the assumption that there is no emission. Specifically, $S_0 \,|\, C \,|\, \overline{e} \xrightarrow[\mathsf{0}\,|\,\mathsf{0}\,|\,\overline{e}]{\tau}_{\{e,\overline{a}\}} S_0 \,|\, \overline{e}{:}\overline{e}.C_1 \,|\, \overline{e}$ will block because $\{e, \overline{a}\} \cap \overline{\mathsf{iA}}^{\tau}(\mathsf{0} \,|\, \mathsf{0} \,|\, \overline{e}) \neq \{\,\}$. Instead, the transition $S_0 \,|\, C \,|\, \overline{e} \xrightarrow[\mathsf{0}\,|\,C\,|\,\mathsf{0}]{\tau}_{\{\overline{e}\}} S_1 \,|\, C \,|\, \mathsf{0}$ which emits the signal can proceed.

(2) Second, $\overline{a} \in \{e, \overline{a}\}$ indicates that the sending prefix $\overline{a}{:}\overline{a}$ of the reader C (which has entered into the synchronisation $\tau = a \,|\, \overline{a}$) must not be consumed by another competing receiver. Specifically, $S_0 \,|\, C \,|\, a \xrightarrow[\mathsf{0}\,|\,\mathsf{0}\,|\,a]{\tau}_{\{e,\overline{a}\}} S_0 \,|\, \overline{e}{:}\overline{e}.C_1 \,|\, a$ will block.

Now, take $S_0 \,|\, C \,|\, D$ where the signal S_0 is connected with two readers, where C is as above and $D = \overline{a}{:}\overline{a}.\overline{e}{:}\overline{e}.D_1$ analogous. Both readers can go ahead and synchronise in reactions $(\tau = a \,|\, \overline{a})$ $S_0 \,|\, C \,|\, D \xrightarrow[\mathsf{0}\,|\,\mathsf{0}\,|\,D]{\tau}_{\{e,\overline{a}\}} S_0 \,|\, \overline{e}{:}\overline{e}.C_1 \,|\, D$ and $S_0 \,|\, C \,|\, D \xrightarrow[\mathsf{0}\,|\,C\,|\,\mathsf{0}]{\tau}_{\{e,\overline{a}\}} S_0 \,|\, C \,|\, \overline{e}{:}\overline{e}.D_1$ which are both strongly enabled since $\overline{\mathsf{iA}}^{\tau}(\mathsf{0} \,|\, \mathsf{0} \,|\, D) = \{a\} = \overline{\mathsf{iA}}^{\tau}(\mathsf{0} \,|\, C \,|\, \mathsf{0})$ which is disjoint from the blocking set $\{e, \overline{a}\}$. This may seem ok, as we wanted to have multiple readers access the signal on the same port a. However, now the emission of $\overline{e}$ by the process that advanced first will block the reading of absence by the other which stayed behind. In each case symmetrically, there is only one (strongly enabled) transition left, viz. $S_0 \,|\, \overline{e}{:}\overline{e}.C_1 \,|\, D \xrightarrow[\mathsf{0}\,|\,\mathsf{0}\,|\,D]{\tau}_{\{\overline{e}\}} S_1 \,|\, C_1 \,|\, D$ and $S_0 \,|\, C \,|\, \overline{e}{:}\overline{e}.D_1 \xrightarrow[\mathsf{0}\,|\,C\,|\,\mathsf{0}]{\tau}_{\{\overline{e}\}} S_1 \,|\, C \,|\, D_1$ where the final states $C_1 \,|\, D$ and $C \,|\, D_1$ of the readers are behaviourally different. Thus, we have non-determinism while the confluence is lost.

The core of the problem is that the concurrent contexts $\mathsf{0} \,|\, \mathsf{0} \,|\, D$ and $\mathsf{0} \,|\, C \,|\, \mathsf{0}$, respectively, of the above rendez-vous synchronisations and do not account for the fact that the signal S_0 can be simultaneously reused by the other reader that is not involved in the transition. We could account for this reuse by adding S_0 into the concurrent context as in $S_0 \,|\, C \,|\, D \xrightarrow[S_0\,|\,\mathsf{0}\,|\,D]{\tau}_{\{e,\overline{a}\}}$ $S_0 \,|\, \overline{e}{:}\overline{e}.C_1 \,|\, D$. Now D can interact with S_0 in the concurrent context $S_0 \,|\, \mathsf{0} \,|\, D$. The above transition depends on the assumption that no other thread will send $\overline{e}$. This is not true, because now $e \in \overline{\mathsf{iA}}^{\tau}(S_0 \,|\, \mathsf{0} \,|\, D)$. Thus, putting the shared signal S_0 into the concurrent

environment is enforcing determinism by blocking the strong enabling properties of the above transitions. Unfortunately, the fix is too strong. The presence of S_0 in the concurrent context would block even the single reader in $S_0 \,|\, \overline{a}{:}\overline{a}$. Specifically, the reduction $S_0 \,|\, \overline{a}{:}\overline{a} \xrightarrow[S_0 \,|\, 0]{\tau}\!\!\!\!\!\!\rightarrow_{\{e,a\}} S_0 \,|\, 0$ would block, because $\overline{a} \in \overline{\mathsf{iA}}(S_0) \subseteq \overline{\mathsf{iA}}^{\tau}(S_0 \,|\, 0)$, and so the reduction is not strongly enabled any more. Thus, our fix would not permit any reading of the signal S_0, whatsoever.

The solution is to tighten up the notion of strong enabling and close it under arbitrary non-clock transitions in the concurrent environment rather than just the silent steps. Technically, we define a larger set $\overline{\mathsf{iA}}^{*}(E) \supseteq \overline{\mathsf{iA}}^{\tau}(E)$ of potential labels (Def. 13) that closes under *all* non-clock actions of E. In this case, we do find $e \in \{e, \overline{a}\} \cap \overline{\mathsf{iA}}^{*}(0 \,|\, 0 \,|\, D) \neq \{\,\}$. In this way, diverging reductions are no longer enabled. On the other hand, the reduction $S_0 \,|\, C \xrightarrow[0 \,|\, 0]{\tau}\!\!\!\!\!\!\rightarrow_{\{e,a\}} S_0 \,|\, \overline{e}{:}\overline{e}.C_1$ remains enabled, because $\overline{\mathsf{iA}}^{*}(0 \,|\, 0) = \{\,\}$. In general, single readers $S_0 \,|\, C$ where C is an arbitrary sequential process, or multiple readers $S_0 \,|\, C \,|\, D$ where *at most one* of C or D is writing, i.e., sending $\overline{e}$, remain enabled.

The last two examples show the policy conformance of Esterel signals presented in Sec. 11 and another ABRO encoding using multi-cast.

▸ **Example 38** (On Esterel signals, reloaded)**.** A signal has the sort $\mathcal{L}(S_v) = \{emit, pres, abs, \sigma\}$ and is conformant to the policy psig with $\mathsf{psig} \Vdash emit \dashrightarrow abs$ and $\mathsf{psig} \Vdash \ell \dashrightarrow \sigma$ for all $\ell \in \{emit, pres, abs\}$.

▸ **Example 39** (On Fully Multi-Cast ABRO)**.** To model full multi-cast ABRO in $\mathsf{CCS}^{\mathsf{spt}}$, we could use sharable (i.e., non-self-blocking) prefixes on internal actions to show how they are useful to model the control-flow of Esterel programs, specifically termination and reset.

- The reset watchdog R can receive a reset signal r or silently move to state $\sigma.R$ where it synchronises on the clock σ to repeat in the next macro-step. When a reset r is received by the environment, then R executes the sequence $r{:}r.\overline{k}_1{:}\overline{k}_1.\overline{k}_2{:}\overline{k}_2.\overline{k}_4{:}\overline{k}_4.\overline{k}_5{:}\overline{k}_5.\mathsf{ABRO}$ whereby it sends a "kill" signal $\overline{k}_i$ to each of the four threads A, B, T and O, in some arbitrary but fixed order. When all kills have been delivered, the watchdog R will restart ABRO. Until such time, there is no clock possible. The second transition $R \xrightarrow[0]{\tau}\!\!\!\!\!\!\rightarrow_{\{r\}} \sigma.\mathsf{R}$ preempts this reset choice but has lower priority as it is blocked by r. It is only taken when there is no reset possible, i.e., when the reset signal r is absent. Note that with the preemption of the the kill signals $\overline{k}_i$ are removed from the potential, because $\overline{\mathsf{iA}}^{\tau}(\sigma.R) = \{\,\}$, i.e., we do not look past the clock prefix.
- The A and B processes are analogous to each other with the role of a and b swapped. Looking at A, we see that it has a choice of three actions, viz to receive a kill signal k_1, an input signal a or perform the clock action σ to start the next macro step. The kill k_1 has highest priority, action a second and the clock can only go ahead if there is neither a k_1 nor an a signal to be received. When there is no kill but an a signal, then the process A sends an $\overline{s}$ signal to indicate its termination. When the kill k_1 occurs, it enters into the halt state 1 where it is inactive but engages in any number of clock ticks. Since this is the neutral element for parallel composition, A essentially drops out of the picture. Note that we cannot use the inactive 0 here, because then the terminated process A would block the clock for all other threads. Finally, when there is neither a kill k_1 nor signal a, the A process can synchronise on σ and repeat as A in the next macro-step. In this fashion, A will patiently wait for a clock tick in which k_1 or a occur. If both occur, the kill takes priority: For instance, the transition $\mathsf{A} \,|\, \overline{k}_1.1 \,|\, \overline{a}.1 \xrightarrow[0 \,|\, 0 \,|\, \overline{a}.1]{\tau}\!\!\!\!\!\!\rightarrow_{\{k_1\}} 1 \,|\, 1 \,|\, \overline{a}.1$ is constructively enabled while $\mathsf{A} \,|\, \overline{k}_1.1 \,|\, \overline{a}.1 \xrightarrow[0 \,|\, \overline{k}_1.1 \,|\, 0]{\tau}\!\!\!\!\!\!\rightarrow_{\{k_1,a\}} 1 \,|\, \overline{k}_1.1 \,|\, 1$ is not enabled.

- The process T is the termination detector implementing the principle discussed above. It forms a clock-patient, killable counter that waits for any possible reception of signal s before it sends termination $\overline{t}$ to process O. Note that T itself does not need to know how many possible termination signals there are. It waits for the termination count (implemented by iA^*) to reach zero.
- When process O receives the termination signal t it sends the output signal $\overline{o}$ to the environment and then halts. Alternatively, with highest priority, it can be killed by reception of k_5 or engage in the clock σ if neither a kill k_5 nor a termination t is present. The reduction $\tau = \overline{t}{:}\{k_4, \overline{t}, s\} \,|\, t{:}\{k_5, t\}$ between T and O will be blocked by precedence, for as long as at least one of the processes A or B has a choice term that sends $\overline{s}$, i.e., as long as $s \in \overline{\mathsf{iA}}^{\tau}(\mathsf{A} \,|\, \mathsf{B})$. Only when both prefix sequences $a{:}\{a, k_1\}.\overline{s}{:}\overline{s}.1$ and $b{:}\{b, k_2\}.\overline{s}{:}\overline{s}.1$ have been eliminated by reductions, then the termination can occur.

Note that we can integrate the termination test into the output process O if we let O loop on consumption of s rather than T. Similarly, we can use only a single kill signal k shared between A, B, O: The reset process R keeps sending $\overline{k}$ unboundedly often, until all target processes have received a copy and all receptions k have been consumed. At hat point, it takes a clock step to repeat ABRO from the start. This yields the following simpler process:

$$
\begin{aligned}
\mathsf{ABRO} &\stackrel{def}{=} \sigma.(\mathsf{R} \,|\, \mathsf{ABO}) \\
\mathsf{ABO} &\stackrel{def}{=} (\mathsf{A} \,|\, \mathsf{B} \,|\, \mathsf{O}) \setminus \{s, t\} \\
\mathsf{R} &\stackrel{def}{=} r{:}r.\mathsf{R}' + \tau{:}r.\sigma.\mathsf{R} \\
\mathsf{R}' &\stackrel{def}{=} \overline{k}.\mathsf{R}' + \tau{:}\overline{k}.\mathsf{ABRO} \\
\mathsf{A} &\stackrel{def}{=} k{:}k.1 + a{:}\{k, a\}.\overline{s}{:}\overline{s}.1 + \sigma{:}\{k, a\}.\mathsf{A} \\
\mathsf{B} &\stackrel{def}{=} k{:}k.1 + b{:}\{k, b\}.\overline{s}{:}\overline{s}.1 + \sigma{:}\{k, b\}.\mathsf{B} \\
\mathsf{O} &\stackrel{def}{=} k{:}k.1 + s{:}k.\mathsf{O} + t{:}\{k, t, s\}.\overline{o}{:}\overline{o}.1 + \sigma{:}\{k, s, t\}.\mathsf{O}.
\end{aligned}
$$

C Auxiliary Results

Here is a simple observation on the properties of $\mathsf{iA}(P)$ and $\mathsf{iA}^*(P)$ that will be used in the proofs below.

▸ **Lemma 40.** *Let P, Q be processes:*

(1) *$P \xrightarrow[R]{\sigma}_H Q$ implies $R \equiv 0$*

(2) *$\alpha \in iA(P)$ iff $P \xrightarrow{\alpha} Q$.*

(3) *$iA(P) \subseteq iA(P \,|\, Q)$*

(4) *If P is single-threaded, then $P \xrightarrow[R]{\alpha}_H Q$ implies $\alpha \in \mathcal{L}$, $R \equiv 0$ and $\tau \notin H$.*

Proof. All points follows by an inspection of the SOS rules, using the definition of $\mathsf{iA}^{\tau}(P)$. ◂

Lem. 40(1) means that clock actions always run in an empty concurrent environment. Observe that Lem. 40(2) means that the set $\mathsf{iA}(P)$ corresponds to the initial actions of P. The presence of τ in the blocking set of a transition provides information about the set of initial actions of a process. Specifically, in rule (Par_3) the presence of τ in the set $H = race(P \,|\, Q)$ indicates a blocking situation arising from the priorities in P or Q conflicting with the rendevous actions ℓ or $\overline{\ell}$. Thus, the condition $\tau \notin H$ can be used to implement priority-based scheduling. Lem. 40(3) implies that single-threaded processes are never blocked.

▸ **Lemma 41.** *Let P be an arbitrary process:*

(1) *$iA(P) \cap \mathcal{L} \subseteq iA^{\tau}(P) \subseteq iA^*(P)$*

(2) *If* $P \equiv Q$ *then* $iA^*(P) = iA^*(Q)$
(3) *If* $P \xrightarrow{\alpha} Q$ *for* $\alpha \in \mathcal{R} \cup \{\tau\}$, *then* $iA^*(Q) \subseteq iA^*(P)$.

Proof. All points follows by an inspection of the SOS rules, using the definition of $\mathsf{iA}^\tau(P)$. ◂

▸ **Definition 42** (Pivot Policy). *A policy* π *is called a* pivot *policy if* $\overline{\pi} \preceq \pi$. *We call a process* P pivotable *if* P *conforms to a pivot policy.*

▸ **Proposition 43.** *A policy* $\pi = (L, \dashrightarrow)$ *is a pivot policy if* $\overline{L} \subseteq L$ *and for all* $\ell_1, \ell_2 \in L$ *with* $\ell_1 \neq \ell_2$ *we have* $\pi \Vdash \ell_1 \diamond \ell_2$ *or* $\pi \Vdash \overline{\ell}_1 \diamond \overline{\ell}_2$.

Proof. Let π be a pivot policy. Since the alphabet of $\overline{\pi}$ is $\overline{L}$ and the alphabet of π is L the assumption $\overline{\pi} \preceq \pi$ implies $\overline{L} \subseteq L$ by definition of $\preceq$. Given labels $\ell_1, \ell_2 \in L$ with $\ell_1 \neq \ell_2$ suppose $\pi \not\Vdash \ell_1 \diamond \ell_2$. Hence $\pi \Vdash \ell_1 \dashrightarrow \ell_2$ or $\pi \Vdash \ell_2 \dashrightarrow \ell_2$. Consider the first case, i.e., $\pi \Vdash \ell_1 \dashrightarrow \ell_2$. Since $\overline{L} \subseteq L$ and $L = \overline{\overline{L}}$ it follows that also $L \subseteq \overline{L}$, i.e., the labels ℓ_i are also in the alphabet of $\overline{\pi}$. But then by definition of the ordering and the fact that $\ell_i = \overline{\overline{\ell}}_i$, the assumption $\overline{\pi} \preceq \pi$ implies that $\overline{\pi} \Vdash \overline{\ell}_1 \dashrightarrow \overline{\ell}_2$. Now, the definition of the dual policy means that $\pi \not\Vdash \overline{\ell}_1 \dashrightarrow \overline{\ell}_2$ and $\pi \not\Vdash \overline{\ell}_2 \dashrightarrow \overline{\ell}_1$. This is the same as $\pi \Vdash \overline{\ell}_1 \diamond \overline{\ell}_2$ as desired.

Suppose the properties (1) and (2) of the proposition hold for a policy π. We claim that $\overline{\pi} \preceq \pi$. The inclusion of alphabets is by assumption (1). Then, suppose $\pi \Vdash \overline{\ell}_1 \dashrightarrow \overline{\ell}_2$ which implies $\pi \not\Vdash \overline{\ell}_1 \diamond \overline{\ell}_2$. If $\ell_1 = \ell_2$ we have $\overline{\pi} \Vdash \overline{\ell}_1 \dashrightarrow \overline{\ell}_2$ directly by definition. So, let $\ell_1 \neq \ell_2$. Because of $\overline{L} \subseteq L$ and (2) this gives us $\pi \Vdash \ell_1 \diamond \ell_2$ considering again that $\ell_i = \overline{\overline{\ell}}_i$. But $\pi \Vdash \ell_1 \diamond \ell_2$ is the same as $\overline{\pi} \Vdash \overline{\ell}_1 \dashrightarrow \overline{\ell}_2$ by definition. ◂

▸ **Proposition 44.** *Suppose* P *is input-scheduled and* $P \xrightarrow[R]{\alpha}_H Q$ *for some* α, H, R, Q. *Then:*

1. *If* $\alpha \in \overline{\mathcal{A}}$ *then* $H \subseteq \{\alpha, \tau\}$
2. *If* $\alpha \in \mathcal{A} \cup \mathcal{C} \cup \{\tau\}$ *then* $H \subseteq \mathcal{A} \cup \mathcal{C} \cup \{\tau\}$.

Proof. Obvious by definition of conformance and the construction of π_{is}. ◂

▸ **Lemma 45.** *Let* $P_1 : \pi$ *and* $P_2 : \pi$ *be coherent for the same pivot policy* π *and* $\ell \in \mathcal{L}$ *such that* $P_1 \xrightarrow[R_1]{\ell}_{H_1} P_1'$ *and* $P_2 \xrightarrow[R_2]{\overline{\ell}}_{H_2} P_2'$. *Then,* $H_2 \cap \overline{\mathsf{iA}}(P_1) \subseteq \{\overline{\ell}\}$ *and* $H_1 \cap \overline{\mathsf{iA}}(P_2) \subseteq \{\ell\}$.

Proof. Let $\ell \in \mathcal{L}$ and the transitions be given as in the statement of the Lemma. Suppose $\alpha \in H_2$ and $\alpha \in \overline{\mathsf{iA}}(P_1) \subseteq \mathcal{L}$, in particular $\alpha \neq \tau$. The former means $\pi \Vdash \alpha \dashrightarrow \overline{\ell}$ by Def. 16(1). Since π is pivot we must thus have $\pi \Vdash \overline{\alpha} \diamond \ell$, specifically $\pi \not\Vdash \overline{\alpha} \dashrightarrow \ell$, i.e., $\overline{\alpha} \notin H_1$, again by Def. 16(1). But since $\overline{\alpha} \in \mathsf{iA}(P_1)$ there is a transition $P_1 \xrightarrow[E]{\overline{\alpha}}_H Q$. Since also $\pi \not\Vdash \ell \dashrightarrow \overline{\alpha}$, it follows $\ell \notin H$ by Def. 16(1). Now both $\overline{\alpha} \notin H_1$ and $\ell \notin H$ mean that the c-actions $\ell{:}H_1[R_1]$ and $\overline{\alpha}{:}H[E]$ are interference-free. Note that $\{\ell, \overline{\alpha}\} \subseteq \mathcal{L}$. Thus, the coherence Def. 16 for P_1, implies $\overline{\alpha} = l$ or $\overline{\alpha} \in \mathsf{iA}(R_1)$. The latter is impossible, because $H_2 \cap \overline{\mathsf{iA}}(R_1) \subseteq H_2 \cap \overline{\mathsf{iA}}(R_1 \,|\, R_2) = \{\,\}$ by assumption. Therefore we have $\alpha = \ell$ as desired. The inclusion $H_1 \cap \overline{\mathsf{iA}}(P_2) \subseteq \{\ell\}$ is obtained by a symmetric argument. ◂

▸ **Lemma 46.** *Let* $P : \pi$ *be coherent for a pivot policy* π. *Then, for every reduction* $P \xrightarrow[E]{\tau}_H P'$, *if* $H \cap \overline{\mathsf{iA}}(E) = \{\,\}$ *then* $\tau \notin H$.

Proof. Let P be coherent for pivotable π and a reduction $P \xrightarrow[E]{\tau}_H P'$ such that $H \cap \overline{\mathsf{iA}}(E) = \{\,\}$ be given. Suppose, by contraposition, $\tau \in H$. Since there are no action prefixes $\tau.P'$ in

$\mathsf{CCS}^{\mathsf{spt}}$, the reduction must arise from a synchronisation of two threads P_1 and P_2 inside P, via rule Par_3, i.e.,

$$\frac{P_1 \xrightarrow[R_1]{\ell}_{H_1} P_1' \quad P_2 \xrightarrow[R_2]{\overline{\ell}}_{H_2} P_2' \quad B = \{\ell \,|\, \overline{\ell} \mid H_2 \cap \overline{\mathsf{iA}}(P_1) \not\subseteq \{\overline{\ell}\} \text{ or } H_1 \cap \overline{\mathsf{iA}}(P_2) \not\subseteq \{\ell\}\}}{P_1 \,|\, P_2 \xrightarrow[R_1 \,|\, R_2]{\ell \,|\, \overline{\ell}}_{H_1 \cup H_2 \cup B} P_1' \,|\, P_2'} \; (Par_3)$$

for some $\ell \in \mathcal{L}$, where we may assume, by induction on the depth of the derivation, that $\tau \notin H_1 \cup H_2$. The assumption $H \cap \overline{\mathsf{iA}}(E) = \{\,\}$ implies that $(H_1 \cup H_2) \cap \overline{\mathsf{iA}}(R_1 \,|\, R_2) = \{\,\}$ at the point of (Par_3). We must show that the synchronisation action τ cannot enter into the blocking set, i.e., $\tau = \ell \,|\, \overline{\ell} \notin B$, i.e., which is the same as $H_2 \cap \overline{\mathsf{iA}}(P_1) \subseteq \{\overline{\ell}\}$ and $H_1 \cap \overline{\mathsf{iA}}(P_2) \subseteq \{\ell\}$. The sub-expressions $P_1 : \pi'$ and $P_2 : \pi'$ are also coherent for a common pivot policy π'. This means we can apply Lem. 45 and 46 to show $B = \{\,\}$ and we are done. ◂

D Proofs

Of Prop. 15. First, non-interference of $\alpha_i{:}H_i[E_i \,|\, R]$ means $\alpha_i \notin H_{3-i}$ by assumption. Since $H_i' \subseteq H_i$, this implies also $\alpha_i \notin H_{3-i}'$. Second, suppose that $\alpha_{3-i} = \tau$ for some $i \in \{1,2\}$. Then, non-interference of $\alpha_1{:}H_1[E_1 \,|\, R]$ and $\alpha_2\overline{H}_2[E_2 \,|\, R]$ gives $\tau \notin H_i$ and $H_i \cap \overline{\mathsf{iA}}^*(E_i \,|\, R) = \{\,\}$. Since by Lem. 40, $\overline{\mathsf{iA}}^*(E_i) \subseteq \overline{\mathsf{iA}}^*(E_i \,|\, R)$, the inclusions $H_i' \subseteq H_i$ preserve this property, i.e., $\tau \notin H_i'$ and $H_i' \cap \overline{\mathsf{iA}}^*(E_i) \subseteq H_i \cap \overline{\mathsf{iA}}^*(E_i \,|\, R) = \{\,\}$. ◂

Of Thm. 17. Let P be coherent and Q a derivative with c-enabled reductions $Q \xrightarrow[R_1]{\tau}_{H_1} Q_1$ and $Q \xrightarrow[R_2]{\tau}_{H_2} Q_2$. If $Q_1 \equiv Q_2$ we are done immediately. Suppose $Q_1 \not\equiv Q_2$. Then c-enabling implies $\tau \notin H_i$ as well as $H_i \cap \overline{\mathsf{iA}}^*(R_i) = \{\,\}$, which implies $H_i \cap (\overline{\mathsf{iA}}^*(R_i) \cup \{\tau\}) = \{\,\}$. But then the c-actions $\tau{:}H_1[R_1]$ and $\tau{:}H_2[R_2]$ are interference-free, whence coherence Def. 16(2) implies there exist H_i' and processes R_i', Q' such that $Q_1 \xrightarrow[R_2']{\tau}_{H_2'} Q'$ and $Q_2 \xrightarrow[R_1']{\tau}_{H_1'} Q'$ and $R_1 \overset{\tau}{\rightsquigarrow} R_1'$ and $R_2 \overset{\tau}{\rightsquigarrow} R_2'$ with $H_i' \subseteq H_i$. Finally, observe that $\overline{\mathsf{iA}}^*(R_i') \subseteq \overline{\mathsf{iA}}^*(R_i)$ by Lem. 41 from which we infer $H_i' \cap (\overline{\mathsf{iA}}^*(R_i') \cup \{\tau\}) \subseteq H_i \cap (\overline{\mathsf{iA}}^*(R_i) \cup \{\tau\})$. Thus, the reconverging reductions $Q_i \xrightarrow[R_{3-i}']{\tau}_{H_{3-i}'} Q'$ are again c-enabled. This was to be shown. ◂

Of Prop. 19. The first part of the statement follows directly from Def. 16(2): The c-actions $\ell{:}H_i[R_i]$ are trivially interference-free. Now if $P_1 \not\equiv P_2$, then coherence gives us a strong environment shifts implying $\ell \in \mathsf{iA}(R_1)$. The second part is because by Lem. 40(1) we have $R_i = 0$ and thus $\mathsf{iA}(R_i) = \{\,\}$ when $\ell \in C$. ◂

Of Lem. 23. Consider an admissible transition of $P_1 \,|\, P_2$ where P_1 and P_2 are single-threaded. Firstly, by Lem. 40, each of the single-threaded processes can only generate c-actions $\ell_i{:}H_i[R_i]$ where $\tau \notin H_i$ and $R_i = 0$. Such transitions are always c-enabled for trivial reasons. Further, any rendez-vous synchronisation of such an $\ell_1{:}H_1[R_1]$ and $\ell_2{:}H_2[R_2]$ generates a c-action $\tau{:}(H_1 \cup H_2 \cup B)[R_1 \,|\, R_2]$ with $R_1 \,|\, R_2 = 0$. Because of Lem. 45, the set

$$B = \{\tau \mid H_1 \cap \overline{\mathsf{iA}}(P_2) \subseteq \{\ell\} \text{ or } H_2 \cap \overline{\mathsf{iA}}(P_1) \subseteq \{\overline{\ell}\}\}$$

must be empty. Thus, the reduction $\ell_1 \,|\, \ell_2$ is c-enabled, too. ◂

Of Prop. 24. The proof proceeds by contradiction. Let $P : \pi$ for pivot policy π. Suppose $\ell \in \mathcal{L}$, $\ell \neq \sigma$ and $P \xrightarrow[0]{\sigma}_H P_1$ and $P \xrightarrow[F]{\ell}_N P_2$ and $P \xrightarrow[G]{\overline{\ell}}_M P_2$. By Prop. 18 we infer $\sigma \in N$ or $\ell \in H$, i.e., Conformance Def. 21 we have $\pi \Vdash \sigma \dashrightarrow \ell$ or $\pi \Vdash \ell \dashrightarrow \sigma$. Hence, $\pi \nVdash \sigma \diamond \ell$. For

the same reason we have $\pi \not\Vdash \sigma \diamond \overline{\ell}$. However, this contradicts the pivot property of π, which requires $\pi \Vdash \sigma \diamond \ell$ or $\pi \Vdash \sigma \diamond \overline{\ell}$. Finally, if $P \xrightarrow{\tau} P'$ then the reduction must arise from a rendez-vous synchronisation inside P, i.e., there is $\ell \in \mathcal{R}$ with $\ell \in \mathsf{iA}(P)$ and $\overline{\ell} \in \mathsf{iA}(P)$. But then $\sigma \in \mathsf{iA}(P)$ is impossible as we have just seen. ◂

D.1 Stop and Prefixes

▸ **Proposition 47.** *The process* $\mathsf{0}$ *is coherent for any policy, i.e.,* $\mathsf{0} : \pi$ *for all* π*.*

Proof. The terminating process $\mathsf{0}$ is locally coherent for trivial reasons, simply because it does not offer any transitions at all. ◂

▸ **Proposition 48.** *Let process* Q *be coherent for* π*. Then, for every action* $\ell \in \mathcal{R}$ *the prefix expression* $\ell{:}H.Q$ *is coherent for* π*, if* $\ell \in H \subseteq \{\ell' \mid \pi \Vdash \ell' \dashrightarrow \ell\}$*.*

Proof. A prefix expression $P = \ell{:}H.Q$ generates only a single transition by rule (*Act*), so that the assumption $P \xrightarrow[E_1]{\alpha_1}_{H_1} Q_1$ and $P \xrightarrow[E_2]{\alpha_2}_{H_2} Q_2$ implies $\alpha_i = \ell$, $H_i = H$, $E_i = \mathsf{0}$ and $Q_i = Q$. It is easy to see that $\ell{:}H.Q$ conforms to π if $H \subseteq \{\beta \mid \pi \Vdash \beta \dashrightarrow \alpha\}$. Since $\alpha_i = \ell \in H = H_i$ and $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ nothing needs to be proved. Finally, note that the only immediate derivative of $\alpha{:}H.Q$ is Q which is coherent for π by assumption. ◂

We will later see (Prop. 52) that a prefix $\ell{:}H.Q$ for $\ell \in \mathcal{R}$ can very well be coherent even if $\ell \notin H$. Here we note that for clock prefixes, the blocking set is only constrained by the policy.

▸ **Proposition 49.** *If* Q *is coherent for* π *and clock* $\sigma \in \mathcal{C}$*, then the prefix expression* $\sigma{:}H.Q$ *is locally coherent for* π*, too, if* $H \subseteq \{\beta \mid \pi \Vdash \beta \dashrightarrow \sigma\}$*.*

Proof. The argument runs exactly as in case of Prop. 48. However, we do not need to require condition $\sigma \in H$ because if $\alpha_i = \sigma$ then $\alpha_1 = \alpha_2$, $Q_1 = Q_2$ and $\{\alpha_1, \alpha_2\} \notin \mathcal{R}$. So, no reconvergence is required. Again, we observe that the only immediate derivative of $\sigma{:}H.Q$ is Q which is coherent for π by assumption. ◂

D.2 Summation

▸ **Proposition 50.** *Let* $P_1 : \pi_1$ *and* $P_2 : \pi_2$ *be coherent and for all pairs of initial transitions* $P \xrightarrow[F_1]{\alpha_1}_{H_1} P_1$ *and* $P \xrightarrow[F_2]{\alpha_2}_{H_2} P_2$ *we have* $\alpha_1 \neq \alpha_2$ *as well as* $\alpha_1 \in H_2$ *or* $\alpha_2 \in H_1$*. Then* $P_1 + P_2$ *is coherent for* π *if* $\pi_1 \subseteq \pi$ *and* $\pi_2 \subseteq \pi$*.*

Proof. Let $P = Q + R$ be given with Q and R locally coherent for π_1 and π_2, respectively and the rest as in the statement of the proposition. Every transition of P is either a transition of Q or of R via rule *Sum*. Conformance to π directly follows from the same property of Q or R, respectively. The proof of coherence Def. 16 is straightforward exploiting that two competing but non-interfering transitions must either be both from Q or both from R. More precisely, suppose $P \xrightarrow[F_1]{\alpha_1}_{H_1} P_1$ and $P \xrightarrow[F_2]{\alpha_2}_{H_2} P_2$ with $\alpha_1 \neq \alpha_2$ or $P_1 \neq P_2$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_i \notin H_{3-i}$. Also, we assume that $\alpha_1{:}H_1[F_1]$ and $\alpha_2{:}H_2[F_2]$ are interference-free. Now if the two transitions of P are by (Sum_1) from Q and by (Sum_2) from R, then $\alpha_1 \in \mathsf{iA}(Q)$ and $\alpha_2 \in \mathsf{iA}(R)$. But then by assumption, $\alpha_1 \neq \alpha_2$ and $\alpha_1 \in H_2$ or $\alpha_2 \in H_1$, which implies that $\alpha_1{:}H_1[F_1]$ and $\alpha_2{:}H_2[E_2]$ are not interference-free. This means, to prove coherence Def. 16(2) we may assume that both transitions of P are either by (Sum_1) from Q or both by (Sum_2) from R. Suppose both reductions of $P = Q + R$ are generated by rule (Sum_1), i.e., $Q \xrightarrow[F_1]{\alpha_1}_{H_1} Q_1$ and $Q \xrightarrow[F_2]{\alpha_2}_{H_2} Q_2$ where the transitions generated by (Sum_1)

are $Q + R \xrightarrow[F_1]{\alpha_1}_{H_1} Q_1$ and $Q + R \xrightarrow[F_2]{\alpha_2}_{H_2} Q_2$. Under the given assumptions, the reconverging transitions of coherence Def. 16(2) are obtained directly from coherence of Q, by induction. The case that both transitions are from R by rule (Sum_2) is symmetrical. Finally, the immediate derivatives P_1 and P_2 of $P_1 + P_2$ are coherent for π, because they are coherent for π_i by assumption and thus also for the common extension $\pi_i \subseteq \pi$. ◂

D.3 Parallel

▸ **Lemma 51.** *Let $Q : \pi$ and $R : \pi$ be such that $\pi \setminus \mathcal{R}$ is a pivot policy. Then $(Q \,|\, R) : \pi$.*

Proof. Let $P = Q \,|\, R$ such that both Q and R are coherent for π where $\pi \setminus \mathcal{R}$ is a pivot policy. For conformance consider a transition $P \xrightarrow[E]{\ell}_H P'$ with $\ell \in \mathcal{L}$ and $\ell' \in H$. First suppose $\ell \notin \mathcal{C}$. Then this transition is not a synchronisation but a transition of either one of the sub-processes Q by (Par_1) or P by (Par_2) with the same blocking set H. In either case, we thus use coherence $Q : \pi$ or $R : \pi$ to conclude $\pi \Vdash \ell' \dashrightarrow \ell$. If $\ell = \sigma$, then the transition in question is a synchronisation of Q and R via rule (Par_3) $Q \,|\, R \xrightarrow[0]{\sigma}_{H_2 \cup N_2 \cup B_2} Q_2 \,|\, R_2$ with $E = 0$, $H = H_2 \cup N_2 \cup B_2$ and $P' = Q_2 \,|\, R_2$ generated from transitions $Q \xrightarrow[0]{\sigma}_{H_2} Q_2$ and $R \xrightarrow[0]{\sigma}_{N_2} R_2$. and $B_2 = \{\ell \,|\, \overline{\ell} \mid H_2 \cap \overline{\mathsf{iA}}(R) \not\subseteq \{\sigma\}$ or $N_2 \cap \overline{\mathsf{iA}}(Q) \not\subseteq \{\sigma\}\}$. By Lem. 45 we have $B_2 = \{\}$. If we now take an arbitrary $\ell' \in H \cap \mathcal{L}$ then $\ell' \in H_2$ or $\ell' \in N_2$. In these cases, we obtain $\pi \Vdash \ell' \dashrightarrow \ell$ by conformance Def. 21 from $Q : \pi$ or $R : \pi$.

In the sequel, we tackle coherence Def. 16. First we observe that the immediate derivatives of $Q \,|\, R$ are always parallel compositions $Q' \,|\, R'$ of derivatives of Q and R. Thus, we may assume that the immediate derivatives $Q' \,|\, R'$ are coherent for π because P and Q coherence is preserved under transitions (i.e., by co-induction). Now suppose $P \xrightarrow[E_1]{\alpha_1}_{H_1} P_1$ and $P \xrightarrow[E_2]{\alpha_2}_{H_2} P_2$ such that $\alpha_1 \neq \alpha_2$ or $P_1 \neq P_2$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_i \notin H_{3-i}$. Moreover, the c-actions $\alpha_1{:}H_1[E_1]$ and $\alpha_2{:}H_2[E_2]$ are interference-free. We argue reconvergence by case analysis on the rules that generate these transitions. These could be (Par_1), (Par_2) or (Par_3). First note that by Lem. 45 we can drop the consideration of the synchronisation action $\ell \,|\, \overline{\ell}$ for the blocking set in (Par_3) altogether:

$$\frac{P \xrightarrow[R_1]{\ell}_{H_1} P' \quad Q \xrightarrow[R_2]{\overline{\ell}}_{H_2} Q'}{P \,|\, Q \xrightarrow[R_1 \,|\, R_2]{\ell \,|\, \overline{\ell}}_{H_1 \cup H_2} P' \,|\, Q'} \; (Par_3)$$

We will use the name (Par_{3a}) to refer to the instance of (Par_3) with $\ell \in \mathcal{R}$ and the name $(Par_{3\sigma})$ for the instance with $\ell \in \mathcal{C}$ and we proceeds by case analysis.

- $\{(Par_1), (Par_2)\} \diamond (Par_{3a})$. We start with the case where one of the reductions to P_i is non-synchronising by (Par_1) or (Par_2) and the second reduction to P_{3-i} is a synchronisation obtained by (Par_{3a}). By symmetry, it suffices to consider the case that the non-synchronising transition is by (Par_1) from Q and the synchronising transition by (Par_{3a}) from R, i.e., $\alpha_1 \notin C$ and $\alpha_2 = \ell \,|\, \overline{\ell}$ and $Q \,|\, R \xrightarrow[F_1 \,|\, R]{\alpha_1}_{H_1} Q_1 \,|\, R$ and $Q \,|\, R \xrightarrow[F_2 \,|\, G_2]{\ell \,|\, \overline{\ell}}_{H_2' \cup N_2} Q_2 \,|\, R_2$ with $E_1 = F_1 \,|\, R$, $E_2 = F_2 \,|\, G_2$, $H_2 = H_2' \cup N_2$, $P_1 = Q_1 \,|\, R$ and $P_2 = Q_2 \,|\, R_2$, where $\ell \in \mathcal{R}$ is a rendez-vous action such that $Q \xrightarrow[F_1]{\alpha_1}_{H_1} Q_1$ and $Q \xrightarrow[F_2]{\ell}_{H_2'} Q_2$ and $R \xrightarrow[G_2]{\overline{\ell}}_{N_2} R_2$. We may safely assume $\alpha_1 \neq \ell \,|\, \overline{\ell} = \tau$ or $Q_1 \,|\, R \neq Q_2 \,|\, R_2$. The third case $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ is excluded since $\alpha_2 = \tau$. Further, suppose the c-actions $\alpha_1{:}H_1[F_1 \,|\, R]$ and $\ell \,|\, \overline{\ell}{:}(H_2' \cup N_2)[F_2 \,|\, G_2]$ are non-interfering. Observe that $\{\alpha_1, \ell \,|\, \overline{\ell}\} \notin \mathcal{R}$ and also $\{\alpha_1, \ell \,|\, \overline{\ell}\} \cap \mathcal{C} = \{\}$. Thus only a weak environment shift is needed. We wish to exploit coherence of Q. The first

observation is that since $\alpha_2 = \tau$, the assumptions on non-interference Def. 14(2) implies that $H_1 \cap \overline{\mathsf{iA}}^*(F_1 \,|\, R) = \{\}$. Since $\ell = \overline{\overline{\ell}} \in \overline{\mathsf{iA}}(R) \cap \mathcal{L} \subseteq \overline{\mathsf{iA}}^*(F_1 \,|\, R)$ by Lem. 40 and Lem. 41, this implies that $\ell \notin H_1$ up front. Next, if $\alpha_1 \neq \ell \,|\, \overline{\ell}$, non-interference Def. 14(1) means $\alpha_1 \notin H_2 = H_2' \cup N_2$ and thus $\alpha_1 \notin H_2'$. The same is true if $\alpha_1 = \ell \,|\, \overline{\ell} = \tau$, but then by non-interference Def. 14(2) we have the non-interference equation

$$(H_2' \cup N_2) \cap (\overline{\mathsf{iA}}^*(F_2 \,|\, G_2) \cup \{\tau\}) = H_2 \cap (\overline{\mathsf{iA}}^*(E_2) \cup \{\tau\}) = \{\}$$

from which $\alpha_1 = \tau \notin H_2'$ follows. This settles the first part of the non-interference property Def. 14(1) for the diverging transitions out of Q.

For the second part of non-interference, suppose $\alpha_1 = \tau$. Then the non-interference assumptionimplies that $H_2' \cap \overline{\mathsf{iA}}^*(F_2 \,|\, G_2) = \{\}$. Now since $\overline{\mathsf{iA}}^*(F_2) \subseteq \overline{\mathsf{iA}}^*(F_2 \,|\, G_2)$ we conclude from this that $H_2' \cap \overline{\mathsf{iA}}^*(F_2) = \{\}$. This completes the proof that the c-actions $\alpha_1{:}H_1[F_1]$ and $\ell{:}H_2'[F_2]$ must be non-interfering in all cases.

Note that we always have $\alpha_1 \neq \ell$ or $\{\alpha_1, \ell\} \subseteq \mathcal{R}$, and in the latter case also $\alpha_1 \notin H_2'$ and $\ell \notin H_1$ from the above. Hence, we can use coherence Def. 16 of Q and obtain processes F_1', F_2', Q' with reconverging transitions $Q_1 \xrightarrow[F_2']{\ell} {}_{H_2''} Q'$ and $Q_2 \xrightarrow[F_1']{\alpha_1} {}_{H_1'} Q'$ such that $H_2'' \subseteq H_2'$ and $H_1' \subseteq H_1$. We now invoke (Par_{3a}) and (Par_1) to obtain reconverging transitions $Q_1 \,|\, R \xrightarrow[F_2' \,|\, G_2]{\ell \,|\, \overline{\ell}} {}_{H_2'' \cup N_2} Q' \,|\, R_2$ and $Q_2 \,|\, R_2 \xrightarrow[F_1' \,|\, R_2]{\alpha_1} {}_{H_1'} Q' \,|\, R_2$ such that $H_1' \subseteq H_1$ and $H_2'' \cup N_2 \subseteq H_2' \cup N_2$. Regarding the concurrent environments, if $\alpha_1 \in \mathcal{R}$ then the coherence of Q guarantees that $F_1 \xrightarrow{\ell} F_1'$ and $F_2 \xrightarrow{\alpha_1} F_2'$ from which we infer $F_2 \,|\, G_2 \xrightarrow{\alpha_1} F_2' \,|\, G_2$ and $F_1 \,|\, R \xrightarrow{\ell \,|\, \overline{\ell}} F_1' \,|\, R_2$. If $\alpha_1 = \tau$ the above guarantee from the coherence of Q is weaker: We only have $F_1 \overset{\ell}{\rightsquigarrow} F_1'$ and $F_2 \overset{\alpha_1}{\rightsquigarrow} F_2'$. In any case we obtain the environment shifts $F_2 \,|\, G_2 \overset{\alpha_1}{\rightsquigarrow} F_2' \,|\, G_2$ and $F_1 \,|\, R \overset{\ell \,|\, \overline{\ell}}{\rightsquigarrow} F_1' \,|\, R_2$, as required.

- $(Par_1) \diamond (Par_2)$: Suppose the two non-interfering transitions of $P = Q \,|\, R$ are by (Par_1) from Q and by (Par_2) from R. Thus, we are looking at transitions $Q \xrightarrow[F_1]{\alpha_1} {}_{H_1} Q_1$ and $R \xrightarrow[F_2]{\alpha_2} {}_{H_2} R_2$ for $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R} \cup \{\tau\}$ combined via (Par_1) and (Par_2), respectively, to generate $Q \,|\, R \xrightarrow[F_1 \,|\, R]{\alpha_1} {}_{H_1} Q_1 \,|\, R$ and $Q \,|\, R \xrightarrow[Q \,|\, F_2]{\alpha_2} {}_{H_2} Q \,|\, R_2$ with $E_1 = F_1 \,|\, R$ and $E_2 = Q \,|\, F_2$. Then we can directly construct reconverging transitions, applying (Par_2) and (Par_1), respectively, for $Q_1 \,|\, R \xrightarrow[Q_1 \,|\, F_2]{\alpha_2} {}_{H_2} Q_1 \,|\, R_2$ and $Q \,|\, R_2 \xrightarrow[F_1 \,|\, R_2]{\alpha_1} {}_{H_1} Q_1 \,|\, R_2$. Obviously, $Q_1 \,|\, R_2$ reduces to $Q_1 \,|\, R_2$ and by (Par_1) and (Par_2) we also have the strong environment shifts $F_1 \,|\, R \xrightarrow{\alpha_2} F_1 \,|\, R_2$ and $Q \,|\, F_2 \xrightarrow{\alpha_1} Q_1 \,|\, F_2$ which completes the claim for we have, in particular, $F_1 \,|\, R \overset{\alpha_2}{\rightsquigarrow} F_1 \,|\, R_2$ and $Q \,|\, F_2 \overset{\alpha_1}{\rightsquigarrow} Q_1 \,|\, F_2$. Notice that in this case we do not need to assume that Q or R are coherent, i.e., the assumption that the c-actions $\alpha_i{:}H_i[E_i]$ are interference-free.
- $(Par_i) \diamond (Par_i)$ for $i \in \{1, 2\}$: Suppose both non-interfering and diverging reductions are by (Par_1) from Q (or symmetrically both from R by (Par_2)) $Q \,|\, R \xrightarrow[F_1 \,|\, R]{\alpha_1} {}_{H_1} Q_1 \,|\, R$ and $Q \,|\, R \xrightarrow[F_2 \,|\, R]{\alpha_2} {}_{H_2} Q_2 \,|\, R$ with $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R} \cup \{\tau\}$ and $P_i = Q_i \,|\, R$, $E_i = F_i \,|\, R$, where $Q \xrightarrow[F_1]{\alpha_1} {}_{H_1} Q_1$ and $Q \xrightarrow[F_2]{\alpha_2} {}_{H_2} Q_2$. with $\alpha_1 \neq \alpha_2$ or $Q_1 \,|\, R \neq Q_2 \,|\, R$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_i \notin H_{3-i}$. This, in particular, implies $Q_1 \neq Q_2$. Further, assume non-interference of $\alpha_i{:}H_i[F_i \,|\, R]$. We obtain non-interference of $\alpha_i{:}H_i[F_i]$ by Prop. 15. We can therefore apply coherence Def. 16 of Q and obtain processes F_1', F_2' and Q' with reconverging transitions $Q_1 \xrightarrow[F_2']{\alpha_2} {}_{H_2'} Q'$ and $Q_2 \xrightarrow[F_1']{\alpha_1} {}_{H_1'} Q'$ with $H_i' \subseteq H_i$ and such that

$$F_1 \overset{\alpha_2}{\rightsquigarrow} F_1' \text{ and } F_2 \overset{\alpha_1}{\rightsquigarrow} F_2' \tag{1}$$

and if $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 \neq Q_2$, more strongly

$$F_1 \xrightarrow{\alpha_2} F_1' \text{ and } F_2 \xrightarrow{\alpha_1} F_2'. \tag{2}$$

Reapplying (Par_1) we construct the transitions $Q_1 \mid R \xrightarrow[F_2' \mid R]{\alpha_2} H_2' \; Q' \mid R$ and $Q_2 \mid R \xrightarrow[F_1' \mid R]{\alpha_1} H_1' \; Q' \mid R$. Obviously, by (Par_1) and (Par_2) we also obtain transitions $F_1 \mid R \overset{\alpha_2}{\rightsquigarrow} F_1' \mid R$ and $F_2 \mid R \overset{\alpha_1}{\rightsquigarrow} F_2' \mid R$ from (1) or, more strongly, $F_1 \mid R \xrightarrow{\alpha_2} F_1' \mid R$ and $F_2 \mid R \xrightarrow{\alpha_1} F_2' \mid R$ from (2) if $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 \mid R \neq Q_2 \mid R$.

(Par_{3a}) $\diamond$ (Par_{3a}): This is the most interesting case in which we are going to exploit the assumption that Q and R are coherent for the same pivot policy π. Without loss of generality we assume $P = Q \mid R$ and both the reductions $Q \mid R \xrightarrow[E_i]{\alpha_i} H_i \cup N_i \; Q_i \mid R_i$ with $P_i = Q_i \mid R_i$ are τ-actions generated by the communication rule (Par_3), i.e. $\alpha_1 = \ell_1 \mid \overline{\ell_1} = \tau = \ell_2 \mid \overline{\ell_2} = \alpha_2$ for actions $\ell_i \in \mathcal{R}$. Since $\alpha_1 = \alpha_2$ and $\{\alpha_1, \alpha_2\} = \{\tau\} \notin \mathcal{R}$ we only need to prove confluence the case that $P_1 \neq P_2$, i.e., $Q_1 \neq Q_2$ or $R_1 \neq R_2$. Moreover, we only need a weak environment shift, because $\{\alpha_1, \alpha_2\} = \{\tau\} \notin \mathcal{R}$ and $\{\alpha_1, \alpha_2\} \cap \mathcal{C} = \{\,\}$. Also, non-interference Def. 14(2) means that $(H_i \cup N_i) \cap (\overline{\mathsf{iA}}^*(E_i) \cup \{\tau\}) = \{\,\}$, i.e.,

$$H_i \cap (\overline{\mathsf{iA}}^*(E_i) \cup \{\tau\}) = \{\,\} \text{ and } N_i \cap (\overline{\mathsf{iA}}^*(E_i) \cup \{\tau\}) = \{\,\}. \tag{3}$$

Thus, we are looking at transitions $Q \xrightarrow[F_i]{\ell_i} H_i \; Q_i$ and $R \xrightarrow[G_i]{\overline{\ell}_i} N_i \; R_i$. We claim that the two c-actions $\ell_i{:}H_i[F_i]$ of Q, and likewise the two c-actions $\overline{\ell}_i{:}N_i[G_i]$ of R, are interference-free. Note that since $\ell_i \neq \tau$, we only need to consider the first part of Def. 14(1). We first show that at least one of these pairs of c-actions must be interference free, because of coherence and conformance to π where $\pi \setminus \mathcal{R}$ is a pivot policy. It will then transpire that the other must be interference-free, too, because of the assumption (3). Obviously, if $\ell_1 = \ell_2$ then both $\ell_i{:}H_i[F_i]$ of Q and $\overline{\ell}_i{:}N_i[G_i]$ are interference-free for trivial reasons. We distinguish two cases depending on whether ℓ_1 and ℓ_2 are identical or not. So, suppose $\ell_1 \neq \ell_2$. Both Q and R are coherent for the same policy π and $\pi \setminus \mathcal{R}$ is a pivot policy. Thus, $\pi \Vdash \ell_1 \diamond \ell_2$ or $\pi \Vdash \overline{\ell}_1 \diamond \overline{\ell}_2$, by Def. 42. Hence, $\ell_i \notin H_{3-i}$ or $\overline{\ell}_i \notin N_{3-i}$ by conformance Def. 21, which means that at least one of the pairs of c-actions $\ell_i{:}H_i[F_i]$ of Q or the two c-actions $\overline{\ell}_i{:}N_i[G_i]$ must be interference-free by policy. Now we argue that if $\ell_1 \neq \ell_2$ and one of the pairs $\ell_i{:}H_i[F_i]$ or $\overline{\ell}_i{:}N_i[G_i]$ is interference-free the other must be, too, and so we can close the diamond for both Q and R. By symmetry, it suffices to run the argument in case that the c-actions $\ell_i{:}H_i[F_i]$ of Q are non-interfering. Then, we have $\{\ell_1, \ell_2\} \subseteq \mathcal{R}$ and $\ell_i \notin H_{3-i}$. Hence, we invoke coherence Def. 16 of Q obtaining processes F_1' and F_2' with transitions (strong environment shift) $F_1 \xrightarrow{\ell_2} F_1'$ and $F_2 \xrightarrow{\ell_1} F_2'$ and a process Q' with transitions $Q_1 \xrightarrow[F_2']{\ell_2} H_2' \; Q'$ and $Q_2 \xrightarrow[F_1']{\ell_1} H_1' \; Q'$. such that $H_i' \subseteq H_i$. This means $\ell_i \in \mathsf{iA}(F_{3-i}) \subseteq \mathsf{iA}(F_{3-i} \mid G_{3-i}) = \mathsf{iA}(E_{3-i}) \subseteq \overline{\mathsf{iA}}^*(E_{3-i})$ and therefore $\overline{\ell}_i \notin N_{3-i}$ because of (3). Thus, as claimed, the c-actions $\overline{\ell}_i{:}N_i[G_i]$ are interference-free, too. Again $\{\overline{\ell}_1, \overline{\ell}_2\} \subseteq \mathcal{R}$ and $\overline{\ell}_i \notin N_{3-i}$ entitles us to exploit coherence Def. 16 for R from which we obtain processes G_1' and G_2' with transitions (again, a strong environment shift) $G_1 \xrightarrow{\overline{\ell}_2} G_1'$ as well as $G_2 \xrightarrow{\overline{\ell}_1} G_2'$ as well as a process R' with $R_1 \xrightarrow[G_2']{\overline{\ell}_2} N_2' \; R'$ and $R_2 \xrightarrow[G_1']{\overline{\ell}_1} N_1' \; R'$. with $N_i' \subseteq N_i$. Finally, we invoke (Par_3) to obtain the re-converging reductions $Q_1 \mid R_1 \xrightarrow[F_2' \mid G_2']{\ell_2 \mid \overline{\ell_2}} H_2' \cup N_2' \; Q' \mid R'$ and $Q_2 \mid R_2 \xrightarrow[F_1' \mid G_1']{\ell_1 \mid \overline{\ell_1}} H_1' \cup N_1' \; Q' \mid R'$ with $H_i' \cup N_i' \subseteq H_i \cup N_i$, and also $F_i \mid G_i \xrightarrow{\ell_{3-i} \mid \overline{\ell_{3-i}}} F_i' \mid G_i'$ as required, since this implies the weaker form $F_i \mid G_i \overset{\tau}{\rightsquigarrow} F_i' \mid G_i'$ of environment shift. It remains to consider the case that $\ell_1 = \ell = \ell_2$. By assumption $P_1 \neq P_2$ and so we must have $Q_1 \neq Q_2$ or $R_1 \neq R_2$. Suppose $Q_1 \neq Q_2$. If $R_1 \neq R_2$ we apply a symmetric argument with the role of ℓ_i and $\overline{\ell}_i$ interchanged. As observed above, the c-actions $\ell_i{:}H_i[F_i]$ are trivially interference-free according to Def. 14, because $\ell_1 = \ell_2$ and $\ell_i \neq \tau$. Then, by coherence Def. 16 applied to Q we conclude that there are processes F_1' and F_2' with transitions $F_1 \xrightarrow{\ell} F_1'$ and $F_2 \xrightarrow{\ell} F_2'$ and a process Q' with transitions (the strong environment shift

arises here because we have $\ell \in \mathcal{R}$ and $Q_1 \neq Q_2$) $Q_1 \xrightarrow[F_2']{\ell} H_2' \; Q'$ and $Q_2 \xrightarrow[F_1']{\ell} H_1' \; Q'$. such that $H_i' \subseteq H_i$. This means that $\ell \in \mathsf{iA}(F_{3-i}) \subseteq \mathsf{iA}(E_{3-i})$ and so $\overline{\ell} \notin N_{3-i}$ because of $N_{3-i} \cap \overline{\mathsf{iA}}^*(E_{3-i}) = \{\,\}$ as per (3). If $R_1 = R_2$ then for each $i \in \{1,2\}$, by coherence Def. 16 (as $\overline{\ell} \notin N_{3-i} \cup \mathcal{C}$) there is a transition $R_i \xrightarrow[G_{3-i}']{\overline{\ell}} N_{3-i}' \; R'$ with $N_i' \subseteq N_i$ and $G_i = G_i'$ or $G_i \xrightarrow{\overline{\ell}} G_i'$. Using these transitions we now recombine $Q_i \,|\, R_i \xrightarrow[F_{3-i}' \,|\, G_{3-i}']{\ell \,|\, \overline{\ell}} H_{3-i}' \cup N_{3-i}' \; Q' \,|\, R'$ with $H_i' \cup N_i' \subseteq H_i \cup N_i$. Note that if $G_i = G_i'$ then $F_i \,|\, G_i \xrightarrow{\ell} F_i' \,|\, G_i = F_i' \,|\, G_i'$ or if $G_i \xrightarrow{\overline{\ell}} G_i'$ then $F_i \,|\, G_i \xrightarrow{\ell \,|\, \overline{\ell}} F_i' \,|\, G_i'$. This means that, in all cases, $F_i \,|\, G_i \overset{\ell \,|\, \overline{\ell}}{\rightsquigarrow} F_i' \,|\, G_i'$ as desired. If $R_1 \neq R_2$ we can apply coherence Def 16 to R in the same way and close the diamond with processes G_1' and G_2' and transitions (again, the strong environment shift arises here because we have $\overline{\ell} \in \mathcal{R}$ and $R_1 \neq R_2$) $G_1 \xrightarrow{\overline{\ell}} G_1'$ and $G_2 \xrightarrow{\overline{\ell}} G_2'$ as well as a process R' with $R_1 \xrightarrow[G_2']{\overline{\ell}} N_2' \; R'$ and $R_2 \xrightarrow[G_1']{\overline{\ell}} N_1' \; R'$. with $N_i' \subseteq N_i$. Finally, we invoke (Par_3) to obtain the reconverging reductions $Q_1 \,|\, R_1 \xrightarrow[F_2' \,|\, G_2']{\ell \,|\, \overline{\ell}} H_2' \cup N_2' \; Q' \,|\, R'$ and $Q_2 \,|\, R_2 \xrightarrow[F_1' \,|\, G_1']{\ell \,|\, \overline{\ell}} H_1' \cup N_1' \; Q' \,|\, R'$ with $H_i' \cup N_i' \subseteq H_i \cup N_i$, and also $F_i \,|\, G_i \xrightarrow{\ell \,|\, \overline{\ell}} F_i' \,|\, G_i'$ which also implies $F_i \,|\, G_i \overset{\ell \,|\, \overline{\ell}}{\rightsquigarrow} F_i' \,|\, G_i'$ as required.

- $(Par_{3a}) \diamond (Par_{3\sigma})$ and, by symmetry, the induction case $(Par_{3\sigma}) \diamond (Par_{3a})$: We assume $P = Q \,|\, R$ and the non-interfering, diverging reductions $Q \,|\, R \xrightarrow[E_1]{\ell \,|\, \overline{\ell}} H_1 \cup N_1 \cup B_1 \; Q_1 \,|\, R_1$ and $Q \,|\, R \xrightarrow[0]{\sigma} H_2 \cup N_2 \cup B_2 \; Q_2 \,|\, R_2$ for $\ell \in \mathcal{L}$, $\sigma \in \mathcal{C}$, with $P_i = Q_i \,|\, R_i$ and $E_i = F_i \,|\, G_i$ are generated by instances (Par_{3a}) and $(Par_{3\sigma})$ of the communication rules, respectively. These transitions arise from $Q \xrightarrow[F_1]{\ell} H_1 \; Q_1$ and $R \xrightarrow[G_1]{\overline{\ell}} N_1 \; R_1$ and $Q \xrightarrow[0]{\sigma} H_2 \; Q_2$ and $R \xrightarrow[0]{\sigma} N_2 \; R_2$. We assume that the c-actions $(\ell \,|\, \overline{\ell})\!:\!(H_1 \cup N_1 \cup B_1)[E_1]$ and $\sigma\!:\!(H_2 \cup N_2 \cup B_2)[\mathsf{0}]$ are non-interfering. Note that $\ell \neq \sigma$ and $\overline{\ell} \neq \sigma$. First, observe that by Lem. 45, $B_1 = \{\,\} = B_2$. Also, $\ell \,|\, \overline{\ell} = \tau \neq \sigma$ and so the non-interference assumption means that $\sigma \notin H_1 \cup N_1 \cup B_1$ as well as $\tau \notin H_2 \cup N_2 \cup B_2$. The other part of the non-interference, from Def. 14(1), does not provide any extra information.

  We claim that $\ell \in H_2$. By contradiction, if $\ell \notin H_2$ then, since $\sigma \notin H_1$, the two c-actions $\ell\!:\!H_1[F_1]$ and $\sigma\!:\!H_2[\mathsf{0}]$ of Q would be interference-free according to Def. 14. But then, since $\{\ell, \sigma\} \cap \mathcal{C} \neq \{\,\}$, coherence Def. 16 applied to Q would imply $\sigma \in \mathsf{iA}(F_1)$ and $\ell \in \mathsf{iA}(\mathsf{0})$. The latter, however, is impossible. Thus, as claimed, $\ell \in H_2$. Likewise, from $\sigma \notin N_1$ we derive $\overline{\ell} \in N_2$ in the same fashion.

  Now we invoke the fact that $\tau \notin B_2$, i.e. $H_2 \cap \overline{\mathsf{iA}}(R) \subseteq \{\sigma\}$ and $N_2 \cap \overline{\mathsf{iA}}(Q) \subseteq \{\sigma\}$. But $\overline{\ell} \in \mathsf{iA}(R)$ and $\ell \in \mathsf{iA}(Q)$, whence we must have $\ell \notin H_2$ and $\overline{\ell} \notin N_2$. This is a contradiction. Hence, transitions obtained by rules Par_{3a} and $Par_{3\sigma}$ are never interference-free.

- $\{(Par_1), (Par_2)\} \diamond (Par_{3\sigma})$: As the representative case, we assume $P = Q \,|\, R$ and the non-interfering, diverging reductions are $Q \,|\, R \xrightarrow[F_1 \,|\, R]{\alpha_1} H_1 \; Q_1 \,|\, R$ and $Q \,|\, R \xrightarrow[0]{\sigma} H_2 \cup N_2 \cup B_2 \; Q_2 \,|\, R_2$ with $\alpha_1 \in \mathcal{R} \cup \{\tau\}$ so that $P_1 = Q_1 \,|\, R$ and $P_2 = Q_2 \,|\, R_2$ are generated by the rule (Par_1) and $(Par_{3\sigma})$, respectively, from transitions $Q \xrightarrow[F_1]{\alpha_1} H_1 \; Q_1$ and $Q \xrightarrow[0]{\sigma} H_2 \; Q_2$ and $R \xrightarrow[0]{\sigma} N_2 \; R_2$. where $\alpha_1 \neq \sigma$ and the c-actions $\alpha_1\!:\!H_1[F_1 \,|\, R]$ and $\sigma\!:\!(H_2 \cup N_2 \cup B_2)[\mathsf{0}]$ are interference-free. First, Def. 14(1) implies $\alpha_1 \notin H_2 \cup N_2 \cup B_2$ and $\sigma \notin H_1$. Further, if $\alpha_1 = \tau$ then $(H_2 \cup N_2 \cup B_2) \cap (\mathsf{iA}^*(\mathsf{0}) \cup \{\tau\}) = \{\,\}$, which in particular means $H_2 \cap (\mathsf{iA}^*(\mathsf{0}) \cup \{\tau\}) = \{\,\}$. Therefore, the c-actions $\alpha_1\!:\!H_1[F_1]$ and $\sigma\!:\!H_2[\mathsf{0}]$ are interference-free. Note that $\{\alpha_1, \sigma\} \cap \mathcal{C} \neq \{\,\}$. Hence we can exploit coherence Def. 16 for Q, obtaining the stronger form of environment shift. But this would imply $\alpha_1 \in \mathsf{iA}(\mathsf{0})$ which is impossible. So we find that transitions obtained by rules Par_i and $Par_{3\sigma}$ are never interference-free in the sense of coherence.

- $(Par_{3\sigma}) \diamond (Par_{3\sigma})$: We assume $P = Q \,|\, R$ and the non-interfering, diverging reductions $Q \,|\, R \xrightarrow{\sigma_1}_{0} {}_{H_1 \cup N_1 \cup B_1} Q_1 \,|\, R_1$ and $Q \,|\, R \xrightarrow{\sigma_2}_{0} {}_{H_2 \cup N_2 \cup B_2} Q_2 \,|\, R_2$ with $P_i = Q_i \,|\, R_i$ and $E_i = F_i \,|\, G_i$ are generated by the communication rule $(Par_{3\sigma})$ and $(Par_{3\sigma})$. Observing that $\{\sigma_1, \sigma_2\} \subseteq \mathcal{R}$ is impossible, here, we may assume $\sigma_1 \neq \sigma_2$ or $Q_1 \,|\, R_1 \neq Q_2 \,|\, R_2$. This can only hold true if $Q_1 \neq Q_2$ or $R_1 \neq R_2$. Furthermore, the two clock transitions of $Q_1 \,|\, Q_2$ can be assumed to be interference-free. From this it follows that the two underlying clock transitions $Q \xrightarrow{\sigma}_{0} {}_{H_1} Q_1$ and $Q \xrightarrow{\sigma}_{0} {}_{H_2} Q_2$ and the likewise the transitions $R \xrightarrow{\sigma}_{0} {}_{N_1} R_1$ and $R \xrightarrow{\sigma}_{0} {}_{N_2} R_2$ are interference-free. If $Q_1 \neq Q_2$ then we could apply coherence of Q and obtain $\sigma \in \mathsf{iA}(0)$, if $R_1 \neq R_2$ the same follows from coherence of R. This is impossible, whence the case can be excluded. ◂

D.4 Repetition

For arbitrary continuation processes A, a regular prefix $a{:}a.A$ needs to be self-blocking to be coherent. Such a prefix cannot receive from two senders, for $a{:}a.A \,|\, \overline{a} \,|\, \overline{a}$ will block under weak enabling. However, if the receiver is willing to engage in action a repeatedly, say if $A \stackrel{def}{=} a.A$ then we can lift the self-blocking. The process $A \,|\, \overline{a} \,|\, \overline{a}$ is coherent and will happily consume both $\overline{a}$ sender actions. Repetition is the universal way of implementing multi-sender and multi-receiver scenarios. Note that we define a new operator, representing recursion via parallel replication on the same channel ℓ, that can be easily encoded in the current syntax.

▸ **Definition 52** (Sequential Bang Prefix). *For every process P, action $\ell \in \mathcal{L}$ and $H \subseteq Act$, let $\ell^*{:}H.P$ be the process defined by the SOS rule*

$$\frac{}{\ell^*{:}H.P \xrightarrow{\ell}_{P} {}_{H} P \,|\, \ell^*{:}H.P} \ (Rep)$$

We use $\ell^*{:}H$ as an abbreviation for $\ell^*{:}H.0$ and ℓ^* for $\ell^*{:}\{\,\}.0$.

The purpose of repetition is to replicate input and output prefixes so they can be consumed multiple times and by multiple threads. To see this, let us notice the difference in the blocking sets of a regular output prefix $\overline{a}{:}\overline{a}.A$ and its repetition $\overline{a}^*.A$. In the former case, the transition $\overline{a}{:}\overline{a}.A \xrightarrow{\overline{a}}_{0} {}_{\{\overline{a}\}} A$ includes the output label in the blocking set $\{\overline{a}\}$, by (Act_1). This reflects the fact that the output prefix $\overline{a}$ is fully consumed by the transition. Accordingly, (strong/weak) enabling will block multiple concurrent receivers trying to access the label a at the same time. Only a single thread can engage in the synchronisation. This is necessary, since the $\overline{a}$ prefix is consumed. The continuation process A may not offer output $\overline{a}$ any more, or if it does, it may be incongruent with A. Therefore, each synchronisation with one receiver thread would preempt another concurrent receiver thread in getting access to $\overline{a}$. For contrast, in a transition $\overline{a}^*.A \xrightarrow{\overline{a}}_{A} {}_{\{\,\}} A \,|\, \overline{a}^*.A$ generated from (Act_1) and (Rep), the blocking set is empty, and so does not prevent concurrent receivers. This is fine since the prefix is not consumed but repeated in the continuation process $A \,|\, \overline{a}^*.A$.

▸ Remark 53 (Why is the standard bang not good enough?). Note that the sequential bang $\ell^*.A$ is not expressible as $\ell^*.A = (\ell.A)^*$ by the standard 'bang' operator P^* of process algebra which satisfies the *false* (hence not present) structural equivalence $P^* \equiv P \,|\, P^*$ in $\mathsf{CCS}^{\mathsf{spt}}$. The corresponding equivalence $\ell^*.A \cong \ell.A \,|\, \ell^*.A$ does not hold in $\mathsf{CCS}^{\mathsf{spt}}$. Let us see why. Suppose the behaviour of ℓ^* is derived from (or identified with that of) $\ell.A \,|\, \ell^*.A$. Then the initial action of $\ell^*.A$ would be $\ell.A \,|\, \ell^*.A \xrightarrow{\ell}_{\ell^*.A} {}_{\{\,\}} A \,|\, \ell^*.A$ generated by rules (Act_1) and (Par_1). Notice that the concurrent environment $\ell^*.A = \ell^*.A$ is not empty and that we have $\ell \in \mathsf{iA}(\ell^*.A)$. Therefore, a parallel composition

$\overline{\ell}{:}\overline{\ell} \,|\, \ell.A \,|\, \ell^*.A \xrightarrow[\ell^*.A]{\tau}_{\{\overline{\ell}\}} A \,|\, \ell^*.A$ would block under (weak/strong) enabling because $\{\overline{\ell}\} \cap \overline{\mathsf{iA}}(\ell^*.A) \neq \{\,\}$. Thus $\ell.A \,|\, \ell^*.A$ would not be able to synchronise even with a single (self-blocking) sender $\overline{\ell}{:}\overline{\ell}$. For contrast, the sequential bang prefix of Def. 52 can serve arbitrarily many receivers $\overline{\ell}{:}\overline{\ell} \,|\, \overline{\ell}{:}\overline{\ell} \,|\, \ell^*.A \xrightarrow[\overline{\ell}{:}\overline{\ell} \,|\, A]{\tau}_{\{\overline{\ell}\}} \overline{\ell}{:}\overline{\ell} \,|\, A \,|\, \ell^*.A \xrightarrow[A \,|\, A]{\tau}_{\{\overline{\ell}\}} A \,|\, A \,|\, \ell^*.A$, provided that $\{\overline{\ell}\} \cap \overline{\mathsf{iA}}^*(A) = \{\overline{\ell}\} \cap \overline{\mathsf{iA}}^*(A) = \{\,\}$. The problem is that the standard bang $\ell^*.A \equiv \ell.A \,|\, \ell^*.A$ is a *parallel* repetition of label ℓ while the sequential bang $\ell^*.A$ offers all ℓ-labels *sequentially* in a single thread. Only the payload A is offered in parallel. This corresponds to an *async* operator that is triggered by label ℓ, then spawns a child thread A and repeates itself in the main thread. ◂

▸ **Proposition 54.** *Let $P : \pi$ be coherent and $\pi \setminus \mathcal{R}$ a pivot policy. Then, for each $\ell \in \pi \setminus \mathcal{R}$, the sequential bang prefix $\ell^*{:}H.P$ is coherent for π if $H \subseteq \{\ell' \mid \pi \Vdash \ell' \dashrightarrow \ell\}$.*

Proof. By definition, the process $\ell^*{:}H.P$ offers only a single initial transition $\ell^*{:}H.P \xrightarrow[P]{\ell}_H P \,|\, \ell^*{:}H.P$ which must be obtained by application of rule *Rep*. The conformance Def. 21 follows exactly as for action prefixes from $\ell \in \pi$ the assumption $H \subseteq \{\ell' \mid \pi \Vdash \ell' \dashrightarrow \ell\}$. Since there are no diverging transitions for $\ell^*{:}H.P$ the only situation to consider for confluence is that $\ell \notin H$. For reconvergence it suffices to observe that the continuation process $P \,|\, \ell^*.P$ offers another transition engaging with ℓ and the same blocking set H: $P \,|\, \ell^*{:}H.P \xrightarrow[P]{\ell}_H P \,|\, P \,|\, \ell^*{:}H.P$. By structural equivalence, $P \overset{\ell}{\rightsquigarrow} 0 \,|\, P$. This weak environment shift is enough to satisfy coherence Def. 16, because of the deterministic transition of a non-clock. Finally, note that since P is coherent for π and coherence for pivot policies closed under parallel composition (Prop. 51), we can assume that the continuation process $P \,|\, \ell^*{:}H.P$ is conformant to π, by co-induction. ◂

D.5 Hiding

▸ **Proposition 55.** *If Q is coherent for π and $\pi \not\Vdash \sigma \dashrightarrow \ell$ for all $\sigma \in L$, then $Q \,/\, L$ is coherent for π.*

Proof. It suffices to prove the proposition for the special case $Q \,/\, \sigma$ of a single clock. We recall the semantic rule for this case:

$$\frac{P \xrightarrow[E]{\alpha}_H Q \quad H' = H - \{\sigma\}}{P \,/\, \sigma \xrightarrow[E \,/\, \sigma]{\alpha \,/\, \sigma}_{H'} Q \,/\, \sigma} \;(\mathit{Hide})$$

where $\sigma \,/\, \sigma = \tau$ and $\alpha \,/\, \sigma = \alpha$ if $\alpha \neq \sigma$. Let $P = Q \,/\, \sigma$ and the two non-interfering and diverging transitions $P \xrightarrow[E_1]{\alpha_1 \,/\, \sigma}_{H_1} P_1$ and $P \xrightarrow[E_2]{\alpha_2 \,/\, \sigma}_{H_2} P_2$. arise by (*Hide*) from transitions $Q \xrightarrow[F_1]{\alpha_1}_{N_1} Q_1$ and $Q \xrightarrow[F_2]{\alpha_2}_{N_2} Q_2$ with $E_i = F_i \,/\, \sigma$, $P_i = Q_i \,/\, \sigma$ and $H_i = N_i - \{\sigma\}$. For coherence Def. 16 we assume that $\alpha_1 \,/\, \sigma \neq \alpha_2 \,/\, \sigma$ or $P_1 \neq P_2$, or $\{\alpha_1 \,/\, \sigma, \alpha_2 \,/\, \sigma\} \subseteq \mathcal{R}$ and $\alpha_i \,/\, \sigma \notin H_{3-i}$. Further, let the c-actions $\alpha_i{:}H_i[E_i]$ be interference-free. Observe that the first case implies $\alpha_1 \neq \alpha_2$ and the second case means $Q_1 \neq Q_2$. Moreover, because of the assumption that $\pi \not\Vdash \sigma \dashrightarrow \alpha_i$ we must have $\sigma \notin N_1 \cup N_2$. But this means $H_i = N_i$. First, suppose that $\alpha_1 = \alpha_2 = \sigma$. But then the assumption $Q_1 \neq Q_2$ contradicts strong determinacy of clocks (Proposition 19). Thus, the actions α_i cannot both be the clock σ that is hidden. The case $\alpha_i = \sigma$ and $\alpha_{3-i} \in Act - \{\sigma\}$, for some fixed $i \in \{1, 2\}$ can also be excluded as follows: By Prop. 18 the clock transition and the non-clock transition must interfere in Q, i.e., $\alpha_i \in N_{3-i}$ or $\alpha_{3-i} \in N_i$. The former $\sigma = \alpha_i \in N_{3-i}$ is impossible because of the above. The latter is outright impossible since then $\alpha_{3-i} \,/\, \sigma = \alpha_{3-i} \in N_i = H_i$ which contradicts the non-interference assumption on the transitions of P.

The only remaining case to handle is that both of the diverging transitions are by labels $\alpha_1, \alpha_2 \in Act - \{\sigma\}$ and $\alpha_i \,/\, \sigma = \alpha_i$ for both $i \in \{1,2\}$. We claim that the c-actions $\alpha_i{:}N_i[F_i]$ are interference-free. If $\alpha_1 \neq \alpha_2$ then non-interference assumption directly gives $\alpha_i \notin H_i = N_i$. Next, if for some $i \in \{1,2\}$, $\alpha_i = \alpha_i \,/\, \sigma = \tau$, then the assumptions on P not only give $\tau \notin H_{3-i} = N_{3-i}$ but also $N_{3-i} \cap \overline{\mathsf{iA}}^*(F_{3-i} \,/\, \sigma) = H_{3-i} \cap \overline{\mathsf{iA}}^*(E_{3-i}) = \{\,\}$. Since by assumption on policy conformance $\sigma \notin N_{3-i}$ the latter implies $N_{3-i} \cap \overline{\mathsf{iA}}^*(F_{3-i}) = \{\,\}$. Now we have shown that the diverging transitions of Q are non-interfering we can use the coherence Def. 16 for Q by induction hypothesis and get F_1', F_2', Q' such that $Q_1 \xrightarrow[F_2']{\alpha_2} N_2' \; Q'$ and $Q_2 \xrightarrow[F_1']{\alpha_1} N_1' \; Q'$ with $F_1 \overset{\alpha_2}{\rightsquigarrow} F_1'$ and $F_2 \overset{\alpha_1}{\rightsquigarrow} F_2'$ such that $N_i' \subseteq N_i$. By applying (*Hide*) to these transitions, we obtain $P_1 \xrightarrow[F_2' \,/\, \sigma]{\alpha_2 \,/\, \sigma} H_2'' \; Q' \,/\, \sigma$ and $P_2 \xrightarrow[F_1' \,/\, \sigma]{\alpha_1 \,/\, \sigma} H_1'' \; Q' \,/\, \sigma$ where $H_i'' = N_i' - \{\sigma\} \subseteq N_i - \{\sigma\} = H_i$. Note, by rule (*Hide*) we also have $F_1 \,/\, \sigma \overset{\alpha_2 \,/\, \sigma}{\rightsquigarrow} F_1' \,/\, \sigma$ and $F_2 \,/\, \sigma \overset{\alpha_1 \,/\, \sigma}{\rightsquigarrow} F_2' \,/\, \sigma$. In the special case that $\{\alpha_1 \,/\, \sigma, \alpha_2 \,/\, \sigma\} = \{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 \,/\, \sigma \neq Q_2 \,/\, \sigma$, or $\{\alpha_1 \,/\, \sigma, \alpha_2 \,/\, \sigma\} \cap \mathcal{C} = \{\alpha_1, \alpha_2\} \cap \mathcal{C} \neq \{\,\}$ we obtain the stronger context shifts $F_1 \xrightarrow{\alpha_2} F_1'$ and $F_2 \xrightarrow{\alpha_1} F_2'$ from which we infer $F_1 \,/\, \sigma \xrightarrow{\alpha_2 \,/\, \sigma} F_1' \,/\, \sigma$ and $F_2 \,/\, \sigma \xrightarrow{\alpha_1 \,/\, \sigma} F_2' \,/\, \sigma$. This completes the proof. ◂

D.6 Restriction

Let us next look at the restriction operator. The following definition is relevant for creating coherent processes under action restriction (App. D.6).

▸ **Definition 56.** *A policy π is called* precedence-closed *for $L \subseteq \mathcal{L}$ if $\ell_1 \in L$ and $\pi \Vdash \ell_1 \dashrightarrow \ell_2$ implies $\ell_2 \in L$.*

Again, the fact that P is locally coherent does not imply that $P \setminus L$ is locally coherent. Consider the process $Q \overset{def}{=} (s + a{:}s \,|\, b.\overline{s}) \setminus s$ which is conformant and pivotable. The signal s blocks the action a by priority in the thread $s + a{:}s$. For the communication partner $\overline{s}$ to become active, however, it needs the synchronisation with an external action $\overline{b}$. The transition $Q \xrightarrow[(1 \,|\, b.\overline{s}) \setminus s]{a} \{\,\} \; (b.\overline{s}) \setminus s$ generated by (*Restr*) does not provide enough information in the blocking set H and environment E for us to be able to characterise the environments in which the action a is blocked. For instance, the parallel composition $Q \,|\, \overline{b}$ will internally set the sender $\overline{s}$ free and thus block the a-transition. From the above transition the (Par_3) rule permits $Q \,|\, \overline{b}$ to offer the a-step. This is not right as it is internally blocked and does not commute with the $b \,|\, \overline{b}$ reduction. A simple solution to avoid such problems and preserve confluence is to force restricted signals so they do not block any visible actions. This restriction suffices in many cases.

As we have noted above, CCS corresponds to the unclocked and (blocking) free processes of $\mathsf{CCS}^{\mathsf{spt}}$ (Prop. 26). For these processes, admissible transitions and strongly enabled transitions coincide. The key result of Milner [22] (Chap. 11, Prop. 19) is that confluence is preserved by inaction $\mathsf{0}$, prefixing $\ell.P$ and *confluent composition* defined as $P \mid_L Q \overset{def}{=} (P \,|\, Q) \setminus L$ where $\mathcal{L}(P) \cap \mathcal{L}(Q) = \{\,\}$ and $\overline{\mathcal{L}(P)} \cap \mathcal{L}(Q) \subseteq L \cup \overline{L}$. We will show how confluence for this fragment of CCS processes follows from our more general results on coherence. From now on, for the rest of this section, we assume that all processes are unclocked, i.e., each prefix $\ell{:}H.Q$ has $H \cup \{\ell\} \subseteq \mathcal{A} \cup \overline{\mathcal{A}}$.

Let us call a process P *discrete* if all prefixes occurring in P are of form $\ell{:}\{\ell\}.Q$, i.e., they are self-blocking. Under strong enabling, a discrete process behaves like a CCS process with the restriction that each action ℓ only synchronises if there is a matching partner $\overline{\ell}$ in a *unique* other parallel thread. If the matching partner is not unique, then the scheduling blocks. For instance, $\ell{:}\ell \,|\, \overline{\ell}{:}\overline{\ell} \,|\, \overline{\ell}{:}\overline{\ell}$ blocks because there are two concurrent senders $\overline{\ell}$ matching

the receiver ℓ. Now consider the fragment $\mathsf{CCS}^{\mathsf{cc}}$ of discrete processes built from inaction $\mathsf{0}$, self-blocking prefixes $\ell{:}\{\ell\}.Q$ and confluent composition. Milner's result says that the processes in $\mathsf{CCS}^{\mathsf{cc}}$ are confluent under admissible scheduling.

To emulate Milner's result in our setting, we consider the *sort* $\mathcal{L}(P)$ of a process P as the admissible actions of a *discrete* policy π_P with only reflexive precedences. In other words, $\pi_P \Vdash \ell_1 \dashrightarrow \ell_2$ iff $\ell_1 = \ell_2$. It is easy to see that a discrete process always conforms to π_P. Also, discrete policies are always pivot policies (Def. 42) and dependency-closed for all label sets (Def. 56), as one shows easily. Thus, discrete processes, which are coherent for discrete policies, fullfill all conditions of the preservation laws for inaction $\mathsf{0}$ (Prop. 47), self-blocking action prefix $\ell{:}\{\ell\}.Q$ (Prop. 48), parallel composition $P \,|\, Q$ (Prop. 51) and restriction $P \setminus L$ (Prop. 57). As a consequence and in particular, all processes in $\mathsf{CCS}^{\mathsf{cc}}$ are coherent under strong enabling by our results.

Our final observation now is that, for Milner's fragment $\mathsf{CCS}^{\mathsf{cc}}$, the transition semantics under admissible and strong enabling, as well as the notions of confluence and coherence, coincide. Firstly, if P is discrete then in every c-action $\ell{:}H[E]$ executed by a derivative Q of P we must have $H = \{\ell\}$. Thus, the coherence Def. 16(1) becomes redundant since it is always satisfied. Further, any two c-actions $\ell_i{:}H_i[E_i]$ of transitions $Q \xrightarrow[E_1]{\ell_1}_{H_1} Q_1$ and $Q \xrightarrow[E_2]{\ell_2}_{H_2} Q_2$ for $\ell_1 \neq \ell_2$ or $Q_1 \neq Q_2$ are trivially interference-free. The confluent composition of Milner has the further effect that for every c-action $\ell{:}H[E]$ we must have $\ell \notin \overline{\mathsf{iA}}^*(E)$. This is proven by induction on the structure of $P \in \mathsf{CCS}^{\mathsf{cc}}$. Further, the side condition of Par_3 becomes trivial, and thus generally $\tau \notin H$. This is a result of the side-conditions of the confluent composition. Hence, for $\mathsf{CCS}^{\mathsf{cc}}$, strong enabling coincides with plain admissibility CCS. Notice that the class of discrete processes and the class processes coherent for discrete policies is larger than $\mathsf{CCS}^{\mathsf{CW}}$. In particular, we can explain coherence of broadcast actions with repetitive prefixes. Hence, our results are more general even for the restricted class of discrete behaviours. This will be exploited with the examples discussed in a proper section.

▸ **Proposition 57.** *If $Q : \pi$ and π is precedence-closed for $L \cup \overline{L}$, then $(Q \setminus L) : \pi - L$.*

Proof. It suffices to prove the proposition for the special case $P = Q \setminus a$ for a single channel name $a \in A$. Recall the rule (*Restr*) for this case:

$$\frac{Q \xrightarrow[E]{\alpha}_H R \quad \alpha \notin \{a, \overline{a}\} \quad H' = H - \{a, \overline{a}\}}{Q \setminus a \xrightarrow[E \setminus a]{\alpha}_{H'} R \setminus a} \; (\mathit{Restr})$$

Consider a transition $Q \setminus a \xrightarrow[E \setminus a]{\ell}_{H'} Q' \setminus a$ derived from $Q \xrightarrow[E]{\ell}_H Q'$ with $H' = H - \{a, \overline{a}\}$, $\ell \notin \{a, \overline{a}\}$ and $\ell' \in H'$. Then $\ell' \notin \{a, \overline{a}\}$ and $\ell' \in H$. Conformance Def. 21 applied to Q implies $\pi \Vdash \ell' \dashrightarrow \ell$. But then $\pi - a \Vdash \ell' \dashrightarrow \ell$, too, as both ℓ and ℓ' a distinct from a and $\overline{a}$. This ensures conformance. If $P = Q \setminus a$ and the two non-interfering and diverging transitions $P_2 \xleftarrow[E_2]{\alpha_2}_{H_2} P \xrightarrow[E_1]{\alpha_1}_{H_1} P_1$ arise from rule (*Restr*) then we have diverging transitions $Q_2 \xleftarrow[F_2]{\alpha_2}_{N_2} Q \xrightarrow[F_1]{\alpha_1}_{N_1} Q_1$ with $H_i = N_i - \{a, \overline{a}\}$, $\alpha_i \notin \{a, \overline{a}\}$, where $E_i = F_i \setminus a$ and $P_i = Q_i \setminus a$. We assume $\alpha_1 \neq \alpha_2$ or $P_1 \neq P_2$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_i \notin H_{3-i}$. Further, suppose non-interference of $\alpha_1{:}H_1[E_1]$ and $\alpha_2{:}H_2[E_2]$. Note that in case $P_1 \neq P_2$ we immediately have $Q_1 \neq Q_2$. Next, notice that the premise of the (*Restr*) rule ensures $\alpha_i \notin \{a, \overline{a}\}$. It is easy to see that this means $\{a, \overline{a}\} \cap N_i = \{\,\}$, i.e., $H_i = N_i$. By contraposition, suppose $\ell \in \{a, \overline{a}\}$ and $\ell \in N_i$. Then $\pi \Vdash \ell \dashrightarrow \alpha_i$. But π is precedence-closed for $\{a, \overline{a}\}$, so we would have $\alpha_i \in \{a, \overline{a}\}$ which is a contradiction. So, $H_i = N_i$.

All this means that to invoke coherence Def. 16 of the above Q is sufficent to show that the c-actions $\alpha_1{:}N_1[F_1]$ and $\alpha_2{:}N_2[F_2]$ do not interfere. Now, if $\alpha_1 \neq \alpha_2$ then from the

non-interference assumption on the transitions out of P we get $\alpha_i \notin H_{3-i} = N_{3-i}$. This is what we need for Def. 14(1) to show that the c-actions $\alpha_i{:}N_i[F_i]$ are interference-free. For the second part Def. 14(2) suppose $\alpha_{3-i} = \tau$ for some $i \in \{1,2\}$. Then the assumed non-interference of the transitions out of P means $H_i \cap (\overline{\mathsf{iA}}^*(E_i) \cup \{\tau\}) = \{\}$, which is the same as $\tau \notin H_i$ and the following set condition $(N_i - \{a,\overline{a}\}) \cap \overline{\mathsf{iA}}^*(F_i \setminus a) = \{\}$. Since $\{a,\overline{a}\} \cap N_{3-i} = \{\}$ we have $N_i \cap \overline{\mathsf{iA}}^*(F_i) \subseteq \mathcal{L} - \{a,\overline{a}\}$. We claim that the α_i-transitions of Q is enabled, specifically $N_i \cap \overline{\mathsf{iA}}^*(F_i) = \{\}$ observing that $\tau \notin H_i = N_i$. By contraposition, suppose $\beta \in N_i \cap \overline{\mathsf{iA}}^*(F_i) \subseteq \mathcal{L} - \{a,\overline{a}\}$. Hence, $\beta \notin \{a,\overline{a}\}$ and so $\beta \in N_i - \{a,\overline{a}\}$ and also $\beta \in \overline{\mathsf{iA}}^*(F_i \setminus a)$, but this contradicts the above set-condition. Thus, we have shown that for all $i \in \{1,2\}$, $\alpha_{3-i} = \tau$ implies $N_i \cap \overline{\mathsf{iA}}^*(F_i) = \{\}$. This proves the c-actions $\alpha_i{:}N_i[F_i]$ out of Q are interference-free. Hence, we can use coherence of Q for the diverging transition and get F_1', F_2', Q' $Q_1 \xrightarrow[F_2']{\alpha_2}_{N_2'} Q'$ and $Q_2 \xrightarrow[F_1']{\alpha_1}_{N_1'} Q'$ with $F_1 \overset{\alpha_2}{\rightsquigarrow} F_1'$ and $F_2 \overset{\alpha_1}{\rightsquigarrow} F_2'$ such that $N_i' \subseteq N_i$. We construct the required reconvergence by application of rule (*Restr*): $P_1 \xrightarrow[E_2']{\alpha_2}_{H_2''} Q' \setminus a$ and $P_2 \xrightarrow[E_1']{\alpha_1}_{H_1''} Q' \setminus a$ with $F_1 \setminus a \overset{\alpha_2}{\rightsquigarrow} F_1' \setminus a$ and $F_2 \setminus a \overset{\alpha_1}{\rightsquigarrow} F_2' \setminus a$ with $E_i' = F_i' \setminus a$ and $H_i'' = N_i' - \{a,\overline{a}\} \subseteq N_i - \{a,\overline{a}\} = H_i$. In the special case that $\{\alpha_1,\alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $P_1 \neq P_2$ or $\{\alpha_1,\alpha_2\} \cap \mathcal{C} \neq \{\}$ we can rely on the stronger context shifts $F_1 \xrightarrow{\alpha_2} F_1'$ and $F_2 \xrightarrow{\alpha_1} F_2'$ to conclude $F_1 \setminus a \xrightarrow{\alpha_2} F_1' \setminus a$ and $F_2 \setminus a \xrightarrow{\alpha_1} F_2' \setminus a$. This completes the proof. ◂