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 standard theory of process algebra with priorities, we identify a new scheduling mechanism, called *constructive reduction* which is designed to capture the essence of synchronous programming. The distinctive property of this evaluation strategy is to achieve determinacy-by-construction for multi-cast concurrent communication with shared memory. In the technical setting of CCS extended by clocks and priorities, we prove for a large class of *coherent* processes a confluence property for constructive reductions. We show that under some restrictions, called *pivotability*, coherence is preserved by the operators of prefix, summation, parallel composition, restriction and hiding. Since this permits memory and sharing, we are able to cover a strictly larger class of processes compared to those in Milner's classical confluence theory for 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.

1 Introduction

Concurrency, by expression or by implementation, is both a convenient and unavoidable feature of modern software systems. However, this does not mean that we must necessarily give up the requirement of functional determinism which is crucial for maintaining predictability and to manage design complexity by simple mathematical models [23, 8, 13]. While the pure λ-calculus is naturally deterministic by design, it cannot model shared memory. Process algebras can naturally model shared objects, but do not guarantee determinism out of the box. In this paper we use the standard mathematical formalism CCS [27] to study methods for reconciling concurrency and determinism.

Determinacy in Process Algebra

In CCS, the interaction of concurrent processes $P \,|\, Q$ arises from the rendezvous synchronisation of an *action* α of P and an associated *co-action* $\overline{\alpha}$ from Q, generating a *silent* action τ, also called a *reduction*. A process is in *normal form* if cannot reduce any more and *determinate* if it reduces to at most one normal form, up to some notion of structural congruence. As a simplified example for the scenarios that we are interested in, consider a triple of concurrent processes $R \,|\, S \,|\, W$, where $S \stackrel{def}{=} r.S + w.\overline{r}.0$ acts as a shared resource, such as a (write-once) store, while $R \stackrel{def}{=} \overline{r}.0$, $W \stackrel{def}{=} \overline{w}.0$ are read and write processes, respectively, sharing S with each other. The store S offer a read action r (rendezvous) leaving the state unchanged, or a write action w upon which it changes its state to $\overline{r}.0$. At this point, the store only permits a single read and then becomes the inactive process 0. The synchronisation between the actions r, w and co-actions $\overline{r}$, $\overline{w}$ generates two sequences of reductions from $R \,|\, S \,|\, W$:

$$0 \,|\, \overline{r}.0 \,|\, 0 \xleftarrow{\tau} 0 \,|\, S \,|\, W \xleftarrow{\tau} R \,|\, S \,|\, W \xrightarrow{\tau} R \,|\, \overline{r}.0 \,|\, 0 \xrightarrow{\tau} 0 \,|\, 0 \,|\, 0.$$

Obviously, $R \mid S \mid W$ is not determinate. It reduces to two distinct normal forms $\mathsf{0} \mid \overline{r}.\mathsf{0} \mid \mathsf{0}$ and $\mathsf{0} \mid \mathsf{0} \mid \mathsf{0}$. The final store $\overline{r}.\mathsf{0}$ on the left permits one more read, while $\mathsf{0}$ on the right does not.

A sufficient condition for determinacy is determinism. A process P is *(structurally) deterministic* if for all its derivatives Q and action $\alpha \in Act$, if $Q \xrightarrow{\alpha} Q_1$ and $Q \xrightarrow{\alpha} Q_2$ then $Q_1 \equiv Q_2$, where $\equiv$ is a suitable structural congruence. However, determinism is too strong. For instance, the store $P \stackrel{def}{=} R_1 \mid S \mid R_2$ with two readers $R_i \stackrel{def}{=} \overline{r}.R'_i$ $(i{=}1,2)$ is determinate but not deterministic, when the R_i may reach structurally distinct states after reading, say, $P \xrightarrow{\tau} R'_1 \mid S \mid R_2$ and $P \xrightarrow{\tau} R_1 \mid S \mid R'_2$ where $R'_1 \mid S \mid R_2 \not\equiv R_1 \mid S \mid R'_2$. Moreover, determinism is not closed under parallel composition. In the above example, all process R_1, R_2, S are deterministic, but their composition P is not. An insightful solution, proposed by Milner (Chap. 11 of [27]), is to replace determinism by the notion of *confluence* which still implies determinacy but turns out to be closed under parallel composition, under natural restrictions.

▸ **Definition 1.** *P is (structurally) confluent*[1] *if for every derivative Q of P and reductions $Q \xrightarrow{\alpha_1} Q_1$ and $Q \xrightarrow{\alpha_2} Q_2$ with $\alpha_1 \neq \alpha_2$ or $Q_1 \not\equiv Q_2$, there exist $Q'_1 \equiv Q'_2$ such that $Q_1 \xrightarrow{\alpha_2} Q'_1$ and $Q_2 \xrightarrow{\alpha_1} Q'_2$.*

Milner shows that confluence is preserved by *"confluent composition"* $P \mid_L Q \stackrel{def}{=} (P \mid Q) \setminus L$, combining the parallel and restriction operators, subject to the condition that $\mathcal{L}(P) \cap \mathcal{L}(Q) = \{\,\}$ and $\overline{\mathcal{L}(P)} \cap \mathcal{L}(Q) = L \cup \overline{L}$, where $\mathcal{L}(R)$ is the *sort* of a process R. This is a form of *sort separation*, ensuring that every action α has at most one synchronisation partner $\overline{\alpha}$ either inside or outside of $(P \mid Q) \setminus L$. Many practical examples of determinate systems can be understood as sort-separated compositions of confluent processes. However, this still covers only a rather limited class of applications. Most seriously, memory processes such as S above are intrinsically not confluent, and even if they were, confluent composition forbids direct multi-cast communication, because labels such as r in S could not be shared by two readers R_1, R_2 due to sort-separation. This means that concurrent programming languages that support shared memory and yet have determinate reduction semantics, cannot be handled.

Determinacy in Synchronous Programming

The most prominent class of shared-memory programming languages that manage to reconcile concurrency and determinacy, is known as *Synchronous Programming* (SP). In SP, 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 is defined at the point where all subsystem pause to wait for a global logical *clock*. When this clock arrives, all subsystem proceed into the next computation cycle. Under the so-called *synchrony hypothesis*, the final outcome of a macro-step for each subsystem 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 can be abstracted into a global Mealy automaton with deterministic I/O. SP started with Statecharts [17] and has generated languages such as Signal [14], Lustre [16], Esterel [5], Quartz [37], SCCharts [41], just to mention a few.

When it comes to mathematical modelling it is natural to think of a clock as a broadcast action in the spirit of Hoare's CSP [19] that acts as a synchronisation barrier for a set of concurrent processes. Used with some scheduling control, it can bundle each process' actions

[1] A variation of Milner's "strong confluence" that includes τ-actions and uses $\equiv$ not bisimulation $\sim$.

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 suggests a priority-based scheduling mechanism, i.e., that the clock should fire only if the system has stabilised and no other admissible data actions are possible. This principle, called *maximal progress*, is built into timed process algebras, see e.g. [18, 11].

Contribution

We introduce an extension of CCS, denoted $\mathsf{CCS}^{\mathsf{spt}}$, that brings together the concepts of *priorities* [10, 36, 33] and *clocks* [15, 11, 32] that have previously been studied for CCS, but independently. We thus obtain an adequate compositional setting for SP based on standard techniques from process algebra. This is surprising since many different customised semantics (e.g., [24, 9, 5, 1]) have been developed for SP over the years, few of which fit into the classical framework of CCS. Similar ideas might be possible for other process algebras like ATP [31], TPL [18], CSP or some synchronous variants of Milner's π-calculus [2], just to mention a few. We find CCS most plausible as a point of departure since the concept of priorities is already well established and the combination of asynchronous scheduling and communication via local rendezvous synchronisation makes the problem of ensuring determinacy more prominent.

Our calculus $\mathsf{CCS}^{\mathsf{spt}}$ provides a natural setting to take a fresh look at confluence in CCS as an adequate mathematical model for SP. Moreover, we envisage that $\mathsf{CCS}^{\mathsf{spt}}$, extended by value-passing, can be used as a playground to study compositional embeddings of concurrent λ-calculi. In particular, we have in mind those with sharing, as required for the lazy semantics of Haskell and with deterministic memory such as IVars/TVars [25] and LVars [22]. At the technical level, we make the following specific contributions in this paper:

- We use broadcast actions (clocks) as global synchronisation barriers to schedule processes using priorities in a more flexible way than any of the earlier approaches. While PMC [15] has clocks but no priorities and in CSA [11] the priorities are hardwired and expressing precedence only between the rendezvous actions and the clock, we use a fully general priority scheme as in Phillips' $\mathsf{CCS}^{\mathsf{Ph}}$ [33]. By adding clocks to $\mathsf{CCS}^{\mathsf{Ph}}$ we generalise previous CCS extensions such as Milner's SCCS [26], PMC or CSA for expressing synchronous interactions of concurrent processes. To achieve this, we must strengthen Phillips' notion of *weak enabling* (Def. 2) by *constructive enabling* (Def. 3). While all classical extensions of CCS by priorities [10, 36, 33] schedule the processes by their immediate initial actions, constructive enabling takes into account all actions potentially executable up to the next clock barrier.
- We identify a strictly larger class of processes that enjoy the Church Rosser (CR) Property (and thus reduce to unique normal forms) than the "confluence class" discussed by Milner, see e.g., Chap. 11 of [27]. Since priorities are not part of the classical confluence theory, it cannot model deterministic shared objects with conflicting choice. In this paper, exploiting priorities and clocks, we enrich Milner's notion of confluence to "confluence up-to priorities" that we call *coherence* (Def. 12). We show that coherent processes are CR for constructive enabling (Thm. 14).
- We show that coherence is preserved by parallel composition under reasonable restrictions that permit sharing and memory (Thm. 24). Specifically, we construct coherent processes in a type-directed compositional fashion using the notion of *precedence policies* (Def. 16). The policy for a conformant process specifies an upper bound for its precedence-blocking behaviour (Def. 17). A policy enriches the classical sort of a process by priority information. We define the special class of *pivot policies* and call coherent processes conforming to them

pivotable processes (Def. 48). These are not only coherent (and thus CR) but preserved by parallel composition and other operators of the language (Thm. 24). In this fashion, we are able to extend Milner's classical results for confluent processes to cover significantly more applications, specifically Esterel-style multi-cast SP with shared objects.

• We show that each coherent process is *clock-deterministic* (Prop. 15) and that pivotable processes satisfy *maximal progress* (Prop. 23). No matter in which order we execute constructively enabled reductions, when the normal form is reached, and only then, a clock can tick. Since the normal form is uniquely determined, each pivotable process represents a "synchronous stream". As such, our work extends the classical non-deterministic theory of timed (multi-clocked) process algebras [15, 18, 11] for applications in deterministic synchronous programming.

For lack of space, four separate appendices will present (A) how research towards determinacy for process algebras passes through those above cited papers, (B) a collection of more tricky examples, and (C,D) the full proofs.

2 A Process Algebra with Clocks and Priorities

The syntax and operational semantics of our process algebra 'SynPaTick', denoted $\mathsf{CCS}^{\mathsf{spt}}$, is defined in this section.

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

The terms in $\mathsf{CCS}^{\mathsf{spt}}$ form a set $\mathcal{P}$ of *processes*. We use $P, Q, R, S \ldots$ to range over $\mathcal{P}$. Let $\mathcal{I}$ be a set of *process names* and let $A, B \ldots$ range over $\mathcal{I}$. Let $\mathcal{A}$ be a countably infinite set of *channel names* with $a, b \ldots$ ranging over $\mathcal{A}$. As in CCS, the set $\overline{\mathcal{A}}$ is a disjoint set of *co-names* with elements denoted by $\overline{a}, \overline{b}, \overline{c} \ldots$ in bijection with $\mathcal{A}$; The overbar operator switches between $\mathcal{A}$ and $\overline{\mathcal{A}}$, i.e., $\overline{\overline{a}} = a$ for all $\overline{a} \in \overline{\mathcal{A}}$. We will refer to the 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 $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}} \cup \mathcal{C}$ be the set of input, output and clock *labels* and let ℓ range over $\mathcal{L}$, while L, H range over subsets of $\mathcal{L}$. We write $\overline{L}$ for the set $\{\overline{\ell} \mid \ell \in L\}$. Let $Act = \mathcal{L} \cup \{\tau\}$ be the set of all *actions* obtained by adjoining the *silent* action $\tau \notin \mathcal{L}$ and let α range over 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 *rendezvous actions* to distinguish them from the *clock actions* $\sigma \in \mathcal{C} \subseteq Act$. All symbols can appear indexed. The process terms $\mathcal{P}$ are defined thus:

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

Intuitively, $\mathsf{0}$ denotes the *inactive* process which stops all computations, including clock actions. The *action prefix* $\alpha{:}H.P$ denotes a process that offers to engage in the action α and then behaves 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 rendezvous 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*. We assume that in every restriction $P \setminus L$ the set $L \subseteq \mathcal{R}$ consists of rendezvous actions, and in every hiding P / L we have $L \subseteq \mathcal{C}$. These *scoping* operators act as name binders that introduce local scopes.

$$
\begin{array}{llcl@{\qquad}llcl}
(\mathsf{Comm}) & P \star Q & \equiv & Q \star P & (\mathsf{Idem}_{+}) & P + P & \equiv & P \\
(\mathsf{Assoc}) & (P \star Q) \star R & \equiv & P \star (Q \star R) & (\mathsf{Scope}_{\wr}) & P \wr L \wr L' & \equiv & P \wr (L \cup L') \\
(\mathsf{Zero}) & P \star 0 & \equiv & P & (\mathsf{Scope}_{0}) & 0 \wr L & \equiv & 0
\end{array}
$$

$$
(\mathsf{Scope}_{\alpha}) \quad (\alpha{:}H.P) \setminus L \equiv \begin{cases} 0 & \text{if } \alpha \in L \cup \overline{L} \\ \alpha{:}H.(P \setminus L) & \text{otherwise} \end{cases}
$$

$$
(\mathsf{Scope}_{\setminus}) \quad (P \star Q) \setminus L \equiv \begin{cases} P \star Q \setminus L & \text{if } \mathcal{L}(P) \cap (L \cup \overline{L}) = \{\,\} \\ P \setminus L + Q \setminus L & \text{if } \star = + \\ P \setminus L \,|\, Q \setminus L & \text{if } \star = \,|\, \text{ and } \mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} \cap (L \cup \overline{L}) = \{\,\} \end{cases}
$$

Figure 1 Structural Congruence, where $\star \in \{\,|\,, +\}$ and $\wr \in \{\setminus, /\}$.

The acute reader will notice the absence of the relabelling operator $P[f]$, useful in CCS to define the well-known "standard concurrent form" $(P_1[f_1] \,|\, \cdots \,|\, P_2[f_n]) \setminus L$. 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 $a{:}H.0$. Also, it is useful to drop the braces for the blocking set if it is a singleton, writing $a{:}b.P$ instead of $a{:}\{b\}.P$. In our concrete notation using the abstract syntax of processes, we assume that the binary operator $+$ is right associative and taking higher binding strength than the binary operator $|$, i.e., writing $P + Q + R \,|\, S$ instead of $(P + (Q + R)) \,|\, S$. Also, we assume that the unary operators of prefix, restriction and hiding have higher binding strength than the binary operators. Hence, we write $P \setminus c + a.b.c.Q \,|\, R \,/\, \sigma$ instead of $((P \setminus c) + (a.(b.(c.Q)))) \,|\, (R \,/\, \sigma)$. 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 fragments of $\mathcal{P}$. 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 $Q \,|\, R$. P is *closed* if all sub-processes $\ell{:}H.Q$ in P occur in the scope of a restriction or hiding operator that binds ℓ. As usual, we denote by $\mathcal{L}(P) \subseteq \mathcal{L}$ the free labels, called *(syntactic) sort* of P, identify processes that differ only in the choice of bound labels.

Following Milner's presentation of the π-calculus in [28], we define a structural congruence over processes. Specifically, *structural congruence* $\equiv$ is the smallest congruence over $\mathcal{P}$ that satisfies the equations in Fig. 1. The idea of the structural congruence is to split interaction rules between agents from rules governing the "left-to-right" representation of expressions. 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 transitions that, otherwise, would be blocked. Milner divides CCS equations in static, dynamic and expressions laws. We choose an approach inspired by the work of Berry and Boudol [7] by adding an explicit structural rule in the operational semantics. Note that we do not include in our structural rules all of Milner's τ-laws, i.e. the ones related to the presence of τ-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, Props. 2, 4(2) and 5 of [27]). Since we do not propose notions of behavioural equivalence in this paper, we prefer to take a conservative approach and only consider a minumum set of purely structural rules that do not impinge on the dynamics of expression behaviour.

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) [35, 34], 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 rendezvous 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 decorate the nondeterministic SOS with annotations that provides sufficient information to express a uniform and confluent reduction strategy based on scheduling precedences. 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. The semantics is defined inductively as the smallest relation closed under the rules in Fig. 2. We call each transition derivable by Fig. 2 an *admissible* transition. We denote by $\alpha{:}H[R]$ a compound *c-action* consisting of the three annotations α, H, and 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 refined confluence property. Sometimes, we write $P \xrightarrow{\alpha} Q$ to state that there is a transition for some R and H.

The intuition behind the annotations of admissible transitions is as follows: α is the usual action "passed back" in the SOS. The *blocking set* H is the set of actions that take precedence over α, and R, called the *concurrent context*, is a process that represents the behaviour of all threads in P that execute concurrently with the transition. i.e., all actions in R are potentially in competition with α. Think of H as the resources that are required 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 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. 28).

In the 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 rendezvous actions and "local hiding" of broadcasted clocks. Rules (*Par*) and (*Sum*) are almost standard and deal with parallelism and choice. Rule (*Struct*) internalises in the transition the above presented structural equivalence of processes. Rule (*Com*) constitutes the core of our SOS. It captures, in an elegant way, both the classical CCS rendezvous communication rule *and* the classical CSP broadcasting communication; it has an extra *race* test, enriching the blocking set of the conclusion by τ, in case P and Q interfere. The presence of τ will block unconditionally. Finally, rule (*Con*) deals with process names and refers to a process whose behaviour is specified by a definitional equation $A \stackrel{def}{=} P$.

2.3 Constructive Scheduling

The operational semantics of Fig. 2 for rendezvous actions coincides with that of CCS and the rules for clock actions coincide with the semantics of CSP broadcast actions. The scheduling annotations H and R of a c-action $\alpha{:}H[R]$ expose extra information about blocking and concurrent context, respectively, of the underlying action α. We can use them to implement different scheduling strategies. Prioritised process algebras [10, 36, 33] block an admissible transition $\alpha{:}H[R]$ when H contains the silent action τ or an action that synchronises with some initial action of the concurrent context R. Otherwise, it is (weakly) enabled.

▸ **Definition 2** (Initial actions and weakly-enabled transition)**.**

The set $\mathsf{iA}(R) \subseteq Act$ *of initial actions of a process* R *is given as* $\mathsf{iA}(R) = \{\alpha \mid \exists Q.\, R \xrightarrow{\alpha} Q\}$.

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

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

$$\frac{P \xrightarrow[R]{\alpha}_H P' \quad \alpha \notin \mathcal{C}}{P \,|\, Q \xrightarrow[R \,|\, Q]{\alpha}_H P' \,|\, Q}\ (Par) \qquad \frac{P \xrightarrow[R]{\alpha}_H P'}{P + Q \xrightarrow[R]{\alpha}_H P'}\ (Sum)$$

$$\frac{P \xrightarrow[R_1]{\ell}_{H_1} P' \quad Q \xrightarrow[R_2]{\overline{\ell}}_{H_2} Q' \quad H = race(P,Q)}{P \,|\, Q \xrightarrow[R_1 \,|\, R_2]{\ell \,|\, \overline{\ell}}_{H_1 \cup H_2 \cup H} P' \,|\, Q'}\ (Com) \qquad \frac{A \stackrel{def}{=} P \quad P \xrightarrow[R]{\alpha}_H P'}{A \xrightarrow[R]{\alpha}_H P'}\ (Con)$$

where

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

$$race(P,Q) \stackrel{def}{=} \begin{cases} \{\tau\} & \text{if } H_1 \cap \overline{\mathsf{iA}}(Q) \not\subseteq \{\ell\} \text{ or } H_2 \cap \overline{\mathsf{iA}}(P) \not\subseteq \{\overline{\ell}\} \\ \{\,\} & \text{otherwise} \end{cases}$$

Figure 2 The admissible transitions of $\mathsf{CCS}^{\mathsf{spt}}$ processes. See Def. 2 for the *initial actions* $\mathsf{iA}(P)$.

- *An transition with c-action $\alpha{:}H[R]$ is called weakly enabled if $H \cap (\overline{\mathit{iA}}(R) \cup \{\tau\}) = \{\,\}$.*

Let us call a process P *free* if all its c-actions $\alpha{:}H[R]$ have an empty blocking set $H = \{\,\}$. The weakly enabled transitions of (unclocked) free processes coincide with the semantics of CCS [27]. We call P *discrete* if all its c-actions $\alpha{:}H[R]$ are self-blocking, i.e. $H = \{\alpha\}$. A process P is *irreflexive* if no action prefix $\alpha{:}H.Q$ occurring in P blocks itself, i.e., $\alpha \notin H$. One can show that for (unclocked) irreflexive processes, a transition is weakly enabled iff is derivable in the operational semantics of $\mathsf{CCS}^{\mathsf{Ph}}$ [33][2]. Thus, CCS and $\mathsf{CCS}^{\mathsf{Ph}}$ correspond to theories of free and irreflexive processes of $\mathsf{CCS}^{\mathsf{spt}}$, respectively, under weak enabling. Likewise the theories [10, 36] can be subsumed; specifically, the *prioritised sum* operator $P \mathbin{\mathbin{+}\!\!\!\!\cdot} Q$ of [10] can be coded as $P + Q{:}\mathsf{iA}(P)$ in $\mathsf{CCS}^{\mathsf{spt}}$. The theory [36] proposes two *levels* of labels, $a{:}0$ and $a{:}1$, where 0 denotes higher priority and 1 ordinary priority. Translated into $\mathsf{CCS}^{\mathsf{spt}}$, the prefix $a{:}0.P$ corresponds to $a{:}\{\,\}.P$ while an action prefix $a{:}1.P$ maps in $\mathsf{CCS}^{\mathsf{spt}}$ to $a{:}\mathsf{Prio}.P$ if we define Prio as the set of all high-prioritised labels.

Examples 4 and 5 below in Sec. 3 demonstrate how prioritised scheduling by weak enabling (Def. 2) can indeed help to ensure determinacy in special cases. For the general case, however, weak enabling is not sufficient. In order to prevent non-determinacy (a global property) from priorities (a purely local property), we must be able to block a c-action $\alpha{:}H[R]$ not just if the initial actions $\mathsf{iA}(R)$ of the concurrent context R can synchronise with H, as in weak enabling, but look at the set $\mathsf{iA}^*(R)$ of all actions that R can potentially engage in, before it reaches a clock. Example 7 in Sec. 3 illustrates this.

▸ **Definition 3** (Potential actions and constructive enabling).

- *The set $\mathit{iA}^*(P) \subseteq \mathcal{L}$ of potential actions is the smallest extension $\mathit{iA}(P) \cap \mathcal{L} \subseteq \mathit{iA}^*(P)$ such that if $\ell \in \mathit{iA}^*(Q)$ and $P \xrightarrow{\alpha} Q$ for $\alpha \in \mathcal{R} \cup \{\tau\}$, then $\ell \in \mathit{iA}^*(P)$.*
- *A transition with c-action $\alpha{:}H[R]$ is called constructively enabled, or c-enabled for short, if $H \cap (\overline{\mathit{iA}}^*(R) \cup \{\tau\}) = \{\,\}$.*

[2] In $\mathsf{CCS}^{\mathsf{Ph}}$ the blocking sets are written before the action, $H{:}\alpha.P$, where we write $\alpha{:}H.P$ in $\mathsf{CCS}^{\mathsf{spt}}$.

The following Sec. 3 presents some motivating examples, while Sec. 4 focuses on the mathematical theory underlying $\mathsf{CCS}^{\mathsf{spt}}$.

3 Examples

The first example shows the basic mechanism of how priorities can be used to schedule processes under weak enabling.

▸ **Example 4** (Read Before Write). Consider a "store" $S \stackrel{def}{=} w.r + r{:}w$ that offers a write action w followed by a read r, or a read action r but prefers the write over the read. The fact that w takes priority over r is recorded in the blocking set $\{w\}$ which contains the action w. If we put S in parallel with a reader $R \stackrel{def}{=} \overline{r}$, the read actions r and $\overline{r}$ synchronise in a reduction $S \,|\, R \xrightarrow[\mathsf{0}]{\tau}_{\{w\}} \mathsf{0}$. The blocking set $\{w\}$ will block R's transition when a parallel writer $W \stackrel{def}{=} \overline{w}$ is added, as well. Specifically, the reduction $S \,|\, R \,|\, W \xrightarrow[\overline{w}]{\tau}_{\{w\}} W$ is not weakly enabled, because $\{w\} \cap \overline{\mathsf{iA}}(\overline{w}) = \{w\} \cap \{w\} \neq \{\,\}$. The only enabled reduction is that between S and W giving $S \,|\, R \,|\, W \xrightarrow[\overline{r}]{\tau}_{\{w\}} r \,|\, R$ and only then continue with the read $r \,|\, R \xrightarrow[\mathsf{0}]{\tau}_{\{\}} \mathsf{0}$. Thus, weak enabling makes the reduction determinate. Observe that S is not confluent (Def. 1): We have $S \xrightarrow{w} r$ and $S \xrightarrow{r} \mathsf{0}$ but there is no $P_1 \equiv P_2$ such that $r \xrightarrow{r} P_1$ and $\mathsf{0} \xrightarrow{w} P_2$. However, S is coherent (Def. 12), because the transition $S \xrightarrow[\mathsf{0}]{\tau}_{\{w\}} \mathsf{0}$ interferes (Def. 11) with the former $S \xrightarrow[\mathsf{0}]{w}_{\{\}} r$ as $w \in \{w\}$, whence coherence does not require confluence.

Processes can block each other from making progress. This happens under weak enabling if processes offer two-way rendezvous synchronisation with contradicting precedences.

▸ **Example 5** (Binary Blocking). Take processes $P \stackrel{def}{=} a{:}b.A + b$ and $Q \stackrel{def}{=} \overline{b}{:}\overline{a}.B + \overline{a}$. Their composition $P \,|\, Q$ has two admissible reductions, through $a{:}b \,|\, \overline{a}$ to A or through $b \,|\, \overline{b}{:}\overline{a}$ to B. So, disregarding the precedences, it is non-deterministic if $A \not\equiv B$. Under weak enabling, however, $P \,|\, Q$ creates a circular deadlock: P offers a unless synchronisation with $\overline{b}$ is possible, while Q offers $\overline{b}$ provided there is no synchronisation with a. None of the reductions is enabled. By the "race" side-condition of (Com) the synchronisation $a{:}b \,|\, \overline{a}$ generates the c-action $\tau{:}H[\mathsf{0}]$ with $\tau \in H$ because $\{b\} \cap \overline{\mathsf{iA}}(Q) \not\subseteq \{a\}$. Similarly, since $\{\overline{a}\} \cap \overline{\mathsf{iA}}(P) \not\subseteq \{\overline{b}\}$, the synchronisation $b \,|\, \overline{b}{:}\overline{a}$ is blocked. We note that neither P nor Q is confluent, but they are coherent.

▸ **Example 6** (Reflexive Blocking). Consider the free process $s \,|\, \overline{s}.A \,|\, \overline{s}.B$ where action s can be consumed by $\overline{s}.A$ or by $\overline{s}.B$, generating non-deterministic τ-transitions to $A \,|\, \overline{s}.B$ or to $\overline{s}.A \,|\, B$. However, by adding self-blocking, as in $s{:}s \,|\, \overline{s}.A \,|\, \overline{s}.B$, weak enabling protects the prefix s from being consumed when both $\overline{s}.A$ and by $\overline{s}.B$ compete for it. If only one is present, as in $s{:}s \,|\, \overline{s}.A$ and $s{:}s \,|\, \overline{s}.B$, no blocking occurs.

Blocking under weak enabling ensures determinism in the special case where the circular dependency involves only initial actions. However, parallel threads can create dependency chains through sequences of actions, e.g., when accessing a shared process that is acting as a resource. This is the general case handled by c-enabling.

▸ **Example 7** (Transitive Blocking). 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 $P_0 \,|\, w_0{+}r_0{:}w_0 \,|\, \overline{w_0} \xleftarrow[P_0 \,|\, (w_0+r_0:w_0)]{\tau}_{\{w_1\}} P_0 \,|\, S \,|\, P_1 \xrightarrow[w_1+r_1:w_1 \,|\, P_1]{\tau}_{\{w_0\}} \overline{w_1} \,|\, w_1{+}r_1{:}w_1 \,|\, P_1$.

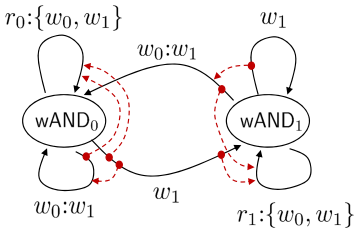
These reductions are both weakly enabled. If we continue under weak enabling then the residual process $\overline{w_1} \,|\, w_1 + r_1{:}w_1 \,|\, P_1$ for the first reduction must first execute the synchronisation on w_1 and thus proceed as $\overline{w_1} \,|\, w_1 + r_1{:}w_1 \,|\, P_1 \xrightarrow{\tau} P_1$ 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 c-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 parallel composition $P_0 \,|\, S \,|\, P_1$ is deterministic under c-enabling. We note again, S is not confluent but coherent (Def. 12).

The next example generalises Ex. 4 to a full memory cell.

▸ **Example 8** (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 $r_v{:}\{w_0, w_1\}$.

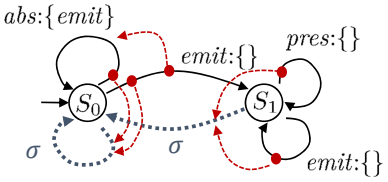


Between the write actions, the blocking sets implement a precedence of w_1 over w_0. In the picture on the right the process state transitions are visualised with the blocking sets as red dashed arrows.

▸ **Example 9** (Esterel Signals)**.** Concurrent Esterel threads communicate via *signals* [6, 38]. 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, 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 are called *non-constructive* and rejected by the compiler. A pure, temporary signal is modelled by the following recursive behaviour:

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

which is depicted on the right and structurally coherent according to Def. 12.



$$
\begin{aligned}
\mathsf{ABRO} &\stackrel{def}{=} \sigma.(\mathsf{A} \mid \mathsf{B} \mid \mathsf{R} \mid \mathsf{O} \mid \mathsf{T}) \setminus \{s,t\} \\
\mathsf{A} &\stackrel{def}{=} k_a{:}k_a.0 + a{:}\{k_a,a\}.\overline{s}{:}\overline{s}.0 + \sigma{:}\{k_a,a\}.\mathsf{A} \\
\mathsf{B} &\stackrel{def}{=} k_b{:}k_b.0 + b{:}\{k_b,b\}.\overline{s}{:}\overline{s}.0 + \sigma{:}\{k_b,b\}.\mathsf{B} \\
\mathsf{R} &\stackrel{def}{=} r{:}r.\overline{k}_a{:}\overline{k}_a.\overline{k}_b{:}\overline{k}_b.\overline{k}_s{:}\overline{k}_s.\overline{k}_t{:}\overline{k}_t.\mathsf{ABRO} + \tau{:}r.\sigma.\mathsf{R} \\
\mathsf{O} &\stackrel{def}{=} k_t{:}k_t.0 + t{:}\{k_t,t\}.\overline{o}{:}\overline{o}.0 + \sigma{:}\{k_t,t\}.\mathsf{O} \\
\mathsf{T} &\stackrel{def}{=} k_s{:}k_s.0 + s{:}\{k_s\}.\mathsf{T} + \overline{t}{:}\{k_s,\overline{t},s\}.0 + \sigma{:}\{k_s,\overline{t},s\}.\mathsf{T}
\end{aligned}
$$

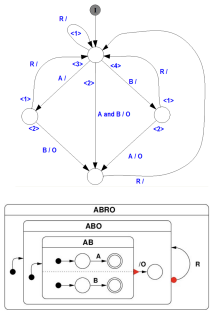


Figure 3 ABRO as a CCS^{spt} process, Mealy machine and SyncChart.

▸ **Example 10** (ABRO). The *hello word* in synchronous languages, see [6], 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. [38]* The behaviour of ABRO is expressed as a Mealy machine and a SyncChart, displayed in Fig. 3 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. In CCS^{spt} we represent the micro-steps as sequences of τ-transitions, i.e., rendez-vous synchronisations, whereas the macro-step is captured as a global clock synchronisation. Unfortunately, the Mealy machine formalism is not scalable because the number of states can grow exponentially as the number of signals grow, while in Esterel can be expressed in a linear number of code lines as seen on the right.

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

The `loop P each R` construct of Esterel executes the behaviour of program `P` for an unbounded number of clock cycles, but restarts its body `P` when the signal `R` becomes present. The reset `R` takes priority over the behaviour of `P` which is preempted at the moment that the reset occurs. Note that in Esterel each signal appears exactly once, as in the specification and unlike in the Mealy machine. Other parts of Esterel also naturally arise from precedence-based scheduling. Esterel's `loop P each R` mechanism, for instance, can be coded using the static parallel composition operator that combines the process `P` to be aborted and reset with a "watchdog" process `R`. The latter waits for the reset signal from the environment in each clock cycle. The choice between the reset and continuing inside `P` is resolved by giving `R` higher priority. When the reset occurs, the watchdog sends "kill" signals to all threads running in `P` and waits for these to terminate. Only then the watchdog `R` restarts `P` from scratch. This is a form of precedence that can be expressed in the blocking sets of CCS^{spt}. As such, Fig. 3 on the left translates the Esterel code of ABRO compositionally in CCS^{spt}. ABRO is obtained as a parallel composition $(\mathsf{R} \mid \mathsf{ABO}) \setminus \{s,t\}$ combining a watchdog R with the behaviour $\mathsf{ABO} \stackrel{def}{=} \mathsf{A} \mid \mathsf{B} \mid \mathsf{O} \mid \mathsf{T}$. 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 $\mathsf{A} \mid \mathsf{B}$ making up the behaviour of the region called AB with code `await A || await B`. The processes $\mathsf{T} \mid \mathsf{O}$ 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. This is seen

in our coding from the fact that the prefixes for these channels are self-blocking.

4 Coherence and Determinacy for CCS^{spt}

This section presents the core elements of our scheduling theory under constructive enabling: coherence, policies, and policy-conformant processes. *Coherence* (Def. 12) corresponds to Milner's notion of confluence, but refined "up-to" priorities. It requires structural confluence for c-enabled and diverging c-actions only if they do not interfere with each other considering their blocking sets and concurrent environments.

▸ **Definition 11** (Transition Interference). *Two c-enabled transitions with c-actions* $\alpha_1{:}H_1[E_1]$ *and* $\alpha_2{:}H_2[E_2]$ *are interference-free if* $\alpha_1 = \alpha_2$ *or* $\alpha_i \notin H_{3-i}$ *for all* $i \in \{1,2\}$.

Observe that for free processes (i.e., with empty blocking sets) all c-actions are interference-free and they are c-enabled iff they are admissible. Confluence of interference-free c-actions is formulated using an auxiliary form of *residual transition* $P \overset{\alpha}{\rightsquigarrow} Q$. This is a transition from P to Q with an action that is a "factor" of α. Formally, we define $P \overset{\alpha}{\rightsquigarrow} Q$ if (i) $\alpha \in \mathcal{L}$ with $P \xrightarrow{\alpha} Q$ or $P \equiv Q$, or (ii) $\alpha = \tau$ with $P \xrightarrow{\tau} Q$ or $P \xrightarrow{\ell} Q$ or $P \xrightarrow{\bar{\ell}} Q$ for some $\ell \in \mathcal{R}$.

▸ **Definition 12** (Process Coherence). *A process* P *is (structurally) coherent if for all its derivatives* Q *the following holds: Given two c-enabled and interference-free 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 both* $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ *and* $\alpha_i \notin H_{3-i}$. *Then, there exist transitions* $Q_1 \xrightarrow[E_2']{\alpha_2}_{H_2'} Q'$ *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* $E_2 \overset{\alpha_1}{\rightsquigarrow} E_2'$.

For free processes, coherence implies confluence (Def. 1), but is strictly stronger.

▸ **Example 13.** For instance, $Q \overset{def}{=} a{:}\{\}.0$ is confluent but not coherent. Since the prefix not self-blocking, coherence ensures that the label must *consumable* by arbitrarily many concurrent processes. Technically, since $Q \xrightarrow{a} Q_1$ and $Q \xrightarrow{a} Q_2$ for $Q_1 \equiv 0 \equiv Q_2$, there should be a transition $Q_i \xrightarrow{a} Q'$ which is not the case. However, for discrete processes, coherence is identical with confluence: With self-blocking, $Q \overset{def}{=} a{:}\{a\}.0$ is both confluent and coherent. It is important to note that all examples in Sec. 3 are coherent but not confluent.

▸ **Theorem 14** (Coherence implies Determinacy). *Every coherent process is determinate under 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. Thm. 14 concerns the τ-transitions of a coherent process. For clock transitions we have an even stronger result.

▸ **Proposition 15** (Clock Determinism). *Every coherent process* P *is structurally clock deterministic, i.e., if* $P \xrightarrow{\sigma} P_1$ *and* $P \xrightarrow{\sigma} P_2$ *then* $P_1 \equiv P_2$.

The question is now how we can obtain coherent processes by composition. In view of Milner's classical result for confluence, we expect to have to impose some form of "sort-separation" for parallel processes. It turns out that we can express such a condition and even avoid the restriction operator (thereby permit sharing) if we enrich the sorts $\mathcal{L}(P)$, $\mathcal{L}(Q)$ by priority information. We call the resulting enriched sorts $\pi(P)$, $\pi(Q)$ *precedence policies*.

▸ **Definition 16** (Precedence 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.*

We will write $\ell \in \pi$ to state that ℓ is in the alphabet of π and we write $\ell_1 \dashrightarrow \ell_2 \in \pi$ to express that the pair (ℓ_1, ℓ_2) is in the precedence relation of the policy. Further, if $\ell_1, \ell_2 \in \pi$ and both $\ell_1 \dashrightarrow \ell_2 \notin \pi$ as well as $\ell_2 \dashrightarrow \ell_1 \notin \pi$, we say that ℓ_1 and ℓ_2 are *concurrently independent* under π, written (by abuse of the set-notation) $\ell_1 \diamond \ell_2 \in \pi$.

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 standard way. There is also the normal inclusion ordering $\pi_1 \subseteq \pi_2$ between policies defined by $L_1 \subseteq L_2$ and $\dashrightarrow_1 \subseteq \dashrightarrow_2$.

The next Def. 17 on conformance captures how policies statically specify the externally observable scheduling behaviour of a process.

▸ **Definition 17** (Conformance). *P conforms to a policy π if for each derivative Q of P: If $Q \xrightarrow[R]{\ell}_H Q'$ with $\ell \in \mathcal{L}$, then $\ell \in \pi$ and for all $\ell' \in H \cap \mathcal{L}$, then $\ell' \dashrightarrow \ell \in \pi$.*

For a process P and policy π, let us write[3] $P \Vdash \pi$ to express that P is coherent and conforms to policy π. We will say that P is *coherent for* π. If $P \Vdash \pi$ then the alphabet of π corresponds to the sort $\mathcal{L}(P)$ in CCS. It gives an upper bound on the labels that can be used by the process, and the precedence relation in π provides a static over-approximation of the blocking, i.e., which labels can be in a choice conflict with each other. As a notational shortcut we will write $P \Vdash \ell_1 \dashrightarrow \ell_2$ to say that there is a policy π with $P \Vdash \pi$ and $\ell_1 \dashrightarrow \ell_2 \in \pi$. Analogously, we write $P \Vdash \ell_1 \diamond \ell_2$ if there is a policy π with $\ell_1 \diamond \ell_2 \in \pi$ and $P \Vdash \pi$.

▸ **Example 18.** The most permissive policy for a given sort $L \subseteq \mathcal{L}$, written $\pi_{max}(L)$, contains all labels from L as its alphabet but no precedences, i.e., $\ell_1 \diamond \ell_2 \in \pi_{max}(L)$ for all $\ell_1, \ell_2 \in L$. If $P \Vdash \pi_{max}$ then P has sort L and will permit arbitrarily many concurrent processes to consume its labels L under c-enabling. The most restrictive policy, denoted $\pi_{min}(L)$, also has alphabet L, but full precedences, i.e., $\ell_1 \dashrightarrow \ell_2 \in \pi_{min}$ for all $\ell_1, \ell_2 \in L$. If $P \Vdash \pi_{min}(L)$, then P has sort L but does not permit any concurrency; it can only communicate with a single sequential thread.

▸ **Example 19.** Reflexive precedences $\ell \dashrightarrow \ell \in \pi$ play a special role in our theory. They permit a conformant $P \Vdash \pi$ and its derivatives to execute ℓ only with a single partner. For instance, ABRO from Ex. 10 satisfies ABRO $\not\Vdash r \diamond r$ and so ABRO $\mid \sigma.\overline{r}$ reduces but ABRO $\mid \sigma.\overline{r} \mid \sigma.\overline{r}$ will block after the initial clock transition. In contrast, if the policy specifies $\ell \diamond \ell \in \pi$, then in a process $P \Vdash \pi$ any ℓ-transition is consumable arbitrarily often. For instance, the Esterel signal from Ex. 9 has $S_1 \Vdash \mathit{pres} \diamond \mathit{pres}$ and so $S_1 \mid \overline{\mathit{pres}} \mid \overline{\mathit{pres}}$ will not block.

Here we are interested in processes that conform to policies with special properties.

▸ **Definition 20** (Pivot and precedence-closed policy). *A policy π is called*

- *pivotable if $\overline{L} \subseteq L$ and for all $\ell_1, \ell_2 \in L$ with $\ell_1 \neq \ell_2$ we have $\ell_1 \diamond \ell_2 \in \pi$ or $\overline{\ell}_1 \diamond \overline{\ell}_2 \in \pi$.*
- *precedence-closed for $L \subseteq \mathcal{L}$ if $\ell_1 \in L$ and $\ell_1 \dashrightarrow \ell_2 \in \pi$ implies $\ell_2 \in L$.*

A process is pivotable/precedence-closed if P conforms to pivot/precedence-closed policy.

[3] In Milner's notation we would write $P : \pi$, but we already use the colon ':' for blocking sets as in $\mathsf{CCS}^{\mathsf{Ph}}$.

▸ **Example 21.** An Esterel signal (Ex. 9) has the alphabet $\{emit, pres, abs\} = \mathcal{L}(S_i)$ ($i \in \{1,2\}$ and for simplicity we ignore the clock σ) and is conformant to the policy psig with $emit \dashrightarrow abs \in \mathsf{psig}$. A typical program R accessing the signal has alphabet $\{\overline{pres}, \overline{abs}\} \subseteq \mathcal{L}(S_i)$ and is of form $R = \overline{pres}{:}\overline{pres}.R_1 + \overline{abs}{:}\{\overline{abs}, \overline{pres}\}.R_0$. If the signal is present then R_1 is executed, if it is absent then R_0 is executed. Such R conforms to a policy prg with $\overline{pres} \dashrightarrow \overline{abs} \in \mathsf{prg}$ and $\ell \dashrightarrow \ell \in \mathsf{prg}$ for $\ell \in \{\overline{pres}, \overline{abs}\}$. Both psig and prg as well as their union $\mathsf{psig} \cup \mathsf{prg}$ are pivot policies, so that $R \,|\, S_i$ is pivotable. In a pivotable process, conflicting choices are resolved by the precedences in a single thread, acting as the "pivot". In $R \,|\, S_i$ the pivot is the signal S_i on reading, resolving the choice $\overline{pres} + \overline{abs}$ in R, and it is the program R on writing, resolving the choice $abs + emit$ in S_i.

Non-pivotable processes, in general, may block because of two threads entering into a circular precedence deadlock with each other.

▸ **Example 22.** For example, recall the processes $P \stackrel{def}{=} a{:}b.A + b$ and $Q \stackrel{def}{=} \overline{b}{:}\overline{a}.B + \overline{a}$ from Example 5. Their parallel composition $P \,|\, Q$ creates a deadlock under (even weak) enabling because of the contradicting precedences in the two possible interactions $a{:}b \,|\, \overline{a}$ and $b \,|\, \overline{b}{:}\overline{a}$. The deadlock is reflected in the policy π of P satisfying $b \dashrightarrow a \in \pi$ and $\overline{a} \dashrightarrow \overline{b} \in \pi$ which is not a pivot policy that would instead require $a \diamond b \in \pi$ or $\overline{a} \diamond \overline{b} \in \pi$.

Reductions of pivotable processes never block because of two threads entering into a circular precedence deadlock with each other. 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 $\mathsf{CCS}^{\mathsf{spt}}$ (c-enabling) all coincide. This is a consequence of the fact (cf. Lem. 51 and 52) that the race test in the rule (Com) is never introducing the silent action τ into the blocking sets. In general, pivotable processes may consist of three or more interacting threads, and blocking may occur and the semantics of CCS, $\mathsf{CCS}^{\mathsf{Ph}}$ and $\mathsf{CCS}^{\mathsf{spt}}$ are different. However, one can show that the blocking of a synchronisation is always due to the external environment. In other words, the computation of $race(P, Q)$ in rule (Com) becomes redundant. Two single-threaded pivotable processes never block each other.

Pivotable process cannot offer a clock action and exhibit reduction at the same time. This means that in this class of processes, clocks and reductions are sequentially scheduled.

▸ **Proposition 23** (Clock Maximal Progress)**.** *Suppose P is coherent and pivotable. If $\sigma \in iA(P)$ then P is in normal form, i.e., there is no reduction $P \xrightarrow{\tau} P'$.*

Prop. 23 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 [18, 11]. More importantly (Thm. 12), 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. The following Thm. 24 identifies a set of closure operators for coherent processes based on policy-conformance. This is our main theorem that summarises Propositions 60–66 (Appendix), which provide in $\mathsf{CCS}^{\mathsf{spt}}$ a generalisation of Milner's Confluence Class, Prop. 15-17 in [27].

▸ **Theorem 24** (Milner's Confluence Class Generalised).

(1) *(Idling)* $\mathsf{0} \Vdash \pi$ *for all policies* π.

(2) *(Rendezvous Prefix) If* $Q \Vdash \pi$, *then for every* $\ell \in \mathcal{R}$ *we have* $\ell{:}H.Q \Vdash \pi$ *if* $\ell \in H \subseteq \{\ell' \mid \ell' \dashrightarrow \ell \in \pi\}$.

(3) *(Clock Prefix) If* $Q \Vdash \pi$ *and clock* $\sigma \in \mathcal{C}$, *then* $\sigma{:}H.Q \Vdash \pi$ *if* $H \subseteq \{\beta \mid \beta \dashrightarrow \sigma \in \pi\}$.

(4) *(Choice) If* $P_1 \Vdash \pi_1$ *and* $P_2 \Vdash \pi_2$ *and for all pairs of initial transitions* $P \xrightarrow[F_1]{\alpha_1}_{H_1} P_1$ *and* $P \xrightarrow[F_2]{\alpha_2}_{H_2} P_2$ *such that* $\alpha_1 \neq \alpha_2$ *as well as* $\alpha_1 \in H_2$ *or* $\alpha_2 \in H_1$. *Then* $P_1 + P_2 \Vdash \pi$ *if* $\pi_1 \subseteq \pi$ *and* $\pi_2 \subseteq \pi$.

(5) *(Parallel) If* $Q \Vdash \pi$ *and* $R \Vdash \pi$ *where* $\pi \setminus \mathcal{R}$ *is a pivot policy, then* $(Q \mid R) \Vdash \pi$.

(6) *(Hiding) If* $Q \Vdash \pi$ *and* $\sigma \dashrightarrow \ell \in \pi$ *for all* $\sigma \in L$ *and* $\ell \in \pi$, *then* $Q \,/\, L \Vdash \pi$.

(7) *(Restriction) If* $Q \Vdash \pi$ *and* π *precedence-closed for* $L \cup \overline{L}$, *then* $(Q \setminus L) \Vdash \pi \setminus L$.

The theorem is the main result of the paper, namely that under a suitable "policy decoration", a process can be run deterministically in the scheduled SOS semantics of $\mathsf{CCS}^{\mathsf{spt}}$.

▸ **Example 25.** To give a concrete example of the property stated by the above theorem, let's revisit Ex. 10: the processes A, B, R, O, T are all coherent and conform to the pivot policy π_{ABRO} with precedences (ignoring the clock, so $\pi \setminus \mathcal{R} = \pi$) $k_s \dashrightarrow \overline{t}$, $s \dashrightarrow \overline{t}$, $k_x \dashrightarrow \{k_x, x\}$ for $x \in \{a, b, s, t\}$ and $\ell \dashrightarrow \ell$ for $\ell \in \{r, a, b, \overline{s}, t, \overline{t}, \overline{o}\}$. Thus, by Thm. 24(5), their composition satisfies $\mathsf{A} \mid \mathsf{B} \mid \mathsf{R} \mid \mathsf{O} \mid \mathsf{T} \Vdash \pi_{\mathsf{ABRO}}$. Since the policy is also precedence-closed for $\{s, t, \overline{s}, \overline{t}\}$, we have $\mathsf{ABRO} \Vdash \pi'_{\mathsf{ABRO}}$ by Thm. 24(3,7), where $\pi'_{\mathsf{ABRO}} = \pi_{\mathsf{ABRO}} \setminus \{s, t\}$.

Note that, specifically, Thm. 24(5) is our replacement of Milner's notion of confluent composition $(P \mid Q) \setminus L$. Finally, to highlight our claim that we extend the classical result, we offer the following definition and conjecture to catch Milner's setting of CCS in $\mathsf{CCS}^{\mathsf{spt}}$.

▸ **Definition 26** (Milner's Confluence Class). *Milner's confluence class* $\mathsf{CCS}^{\mathsf{cc}} \subseteq \mathcal{P}$ *is the set of processes* P *such that for all derivatives* Q *of* P, *if* $Q \xrightarrow[E]{\alpha}_H Q'$, *the following holds:*

- *If* $\alpha \neq \tau$ *then* $H = \{\alpha\}$ *and* $\alpha \notin \overline{iA}^*(E)$.
- *If* $\alpha = \ell \mid \overline{\ell}$ *then* $H = \{\}$ *and* $\{\ell, \overline{\ell}\} \cap \overline{iA}^*(E) = \{\}$.

▸ **Conjecture 1** (Milner's Confluence Class is Coherent). *For all* $P, Q \in \mathsf{CCS}^{\mathsf{cc}}$:

- P *is coherent iff* P *is confluent.*
- *A transition of* P *is c-enabled iff it is admissible.*
- *If* $\mathcal{L}(P) \cap \mathcal{L}(Q) = \{\,\}$ *and* $\overline{\mathcal{L}(P)} \cap \mathcal{L}(Q) = L \cup \overline{L}$, *then* $(P \mid Q) \setminus L \in \mathsf{CCS}^{\mathsf{cc}}$.

5 Further work

We list in spare order some possible future works:

- We plan to study the interaction of our notion of coherence with forms of congruence (static, dynamic and interaction laws in Milner's jargon) and notions of bisimulation.
- We plan to generalise the interaction model of $\mathsf{CCS}^{\mathsf{spt}}$ to synchronisation algebras permitting action fusion in the spirit of Milner's ASCCS and Austry and Boudol's Meije (see [26, 3]).
- Thm. 24 suggests using policies as types. Among possible type disciplines we just mention: behavioural type systems, including session-types to capture rendezvous interactions; intersection-types, including non-idempotent ones to characterise termination and resources awareness, and linear-types to model consuming resources.

- C-actions are somewhat related with continuation passing style, and, in some sense they are more powerful because they permit a "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 in $\mathsf{CCS}^{\mathsf{spt}}$: introducing a partial order could be an interesting, although economic and fine grained, way to reason about time and time priorities.

References

1. J. Aguado, M. Mendler, R. v. Hanxleden, and I. Fuhrmann. Grounding synchronous deterministic concurrency in sequential programming. In Proceedings of the 23rd European Symposium on Programming (ESOP'14), LNCS 8410, pages 229–248, Grenoble, France, April 2014. Springer.
2. R. M. Amadio. A synchronous pi-calculus. Inf. Comput., 205(9):1470–1490, 2007. URL: https://doi.org/10.1016/j.ic.2007.02.002, doi:10.1016/J.IC.2007.02.002.
3. D. Austry and G. Boudol. Algèbre de processus et synchronisation. Theor. Comput. Sci., 30:91–131, 1984.
4. 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.
5. G. Berry. The Constructive Semantics of Pure Esterel. Draft Book, 1999. ftp://ftp-sop.inria.fr/esterel/pub/papers/constructiveness3.ps.
6. G. Berry. The Foundations of Esterel. In Proof, Language and Interaction: Essays in Honour of Robin Milner, pages 425–454. MIT Press, 2000.
7. G. Berry and 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.
8. 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.
9. F. Boussinot and R. De Simone. The SL synchronous language. IEEE Transactions on Software Engineering, 22(4):256–266, 1996.
10. J. Camilleri and G. Winskel. CCS with priority choice. Information and Computation, 116(1):26–37, 1995.
11. 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.
12. V. Dieckert and G. Rozenberg, editors. The Book of Traces. World Scientific, 1995.
13. S. A. Edwards. On determinism. In M. Lohstroh et. al., editor, Lee Festschrift, pages 240–253. Springer, 2018.
14. 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.
15. M. Mendler H. R. Andersen. An asynchronous process algebra with multiple clocks. In D. Sannella, editor, In Proc. ESOP'94, pages 58–73, 1994.
16. 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.
17. 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.
18. M. Hennessy and T. Regan. A process algebra for timed system. Information and Computation, 117:221–239, 1995.
19. C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985. 256 pages.
20. C. D. Kloos and P. Breuer, editors. Formal Semantics for VHDL. Kluwer, 1995.
21. J. W. Klop. Combinatory reduction systems. PhD thesis, Univ. Utrecht, 1980.
22. L. Kuper and R. Newton. Lvars: lattice-based data structures for deterministic parallelism. In Functional High-Performance Computing, page 71–84, New York, NY, USA, 2013. ACM.
23. E. A. Lee. The problem with threads. Computer, 39(5):33–42, 2006.
24. G. Lüttgen, M. von der Beeck, and R. Cleaveland. Statecharts via process algebra. In CONCUR '99: Concurrency Theory, volume 1664 of Lecture Notes in Computer Science, pages 399–414. Springer, 1999.

25 S. Marlow. Parallel and concurrent programming in haskell. In CEFP 2011, pages 339–401, 2012.

26 R. Milner. Calculi for synchrony and asynchrony. Theor. Comput. Sci., 25:267–310, 1983. URL: https://doi.org/10.1016/0304-3975(83)90114-7, doi:10.1016/0304-3975(83)90114-7.

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

28 R. Milner. Communicating and mobile systems - the Pi-calculus. Cambridge University Press, 1999.

29 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.

30 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.

31 X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: theory and application. Inf. Comput., 114(1):131–178, 1994. URL: https://doi.org/10.1006/inco.1994.1083, doi:10.1006/INCO.1994.1083.

32 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.

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

34 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.

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

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

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

38 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.

39 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.

40 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.

41 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 Priorities to Synchronous ones, via Clocks

This rather informal section gives 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.

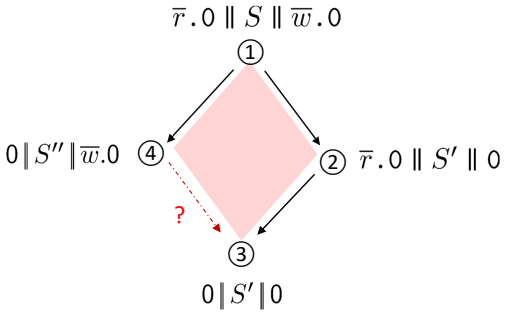


Figure 4 Symmetric Choice. The final outcome, in states ③ and ④, is non-determinate because it depends on the order of read and write. The continuations states are $S' \stackrel{def}{=} r.S'$ and $S'' \stackrel{def}{=} r.S'' + e.S''$. For confluence, the error state ③ should permit a (delayed) read to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

A.1 CCS with (Weak) Priorities

Let us look in more detail on how a precedence mechanism such as "write-before-read" 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 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, 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 is admissible 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 [21] 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. 4. Observe that the behaviour is not only *non-deterministic* in the sense that there is more than one scheduling sequence. It is also non-determinate as the final outcome depends on the scheduling decisions.

Among many CCS extension with priorities, we looked the one of Camilleri and Winskell [10], named $\mathsf{CCS}^{\mathsf{CW}}$ in the sequel. $\mathsf{CCS}^{\mathsf{CW}}$ provides a *prioritised choice* operator $Q \stackrel{def}{=} \alpha_1.Q_1 \nplus \alpha_2.Q_2$ 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 $\mathsf{CCS}^{\mathsf{CW}}$ as $S \stackrel{def}{=} w.S' \dagger r.S''$ in which the prisum $\dagger$ operator enforces a precedence from left to right. The store S is willing to engage in a read action r (to the right of $\dagger$) only if the environment does not offer a write w (on the left of $\dagger$) at the same time, concurrently.

Another important study in the endeavour of adding priorities to CCS is the work of Phillips [33]. In Phillips' extension of CCS with priority guards, 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 [10, 33, 36], 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 (selected) rules for prefix and choice in $\mathsf{CCS}^{\mathsf{CW}}$ are the following[4]:

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

where $\mathsf{iA}(P)$ (defined formally later) informally represents the set of *initial actions* of P that are offered by P in some environment and $\dagger$ 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 $P \dagger Q$ 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 $P \dagger Q$ 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:

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

[4] 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 $R \cap \overline{H} = \{\}$.

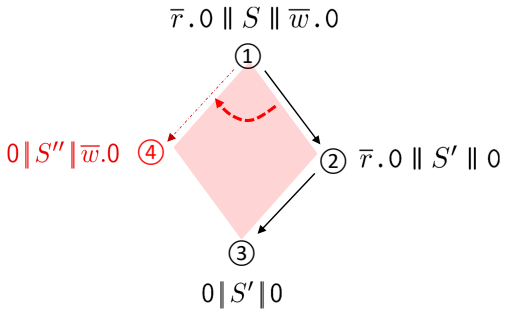


Figure 5 Prioritised Choice. The read-write data race on memory cell S resolved by scheduling priority. The write takes precedence over the read, indicated by the dashed arrow. This removes the top left transition labelled $\tau{:}\{w\}$. Since there is only one execution path, the system is deterministic. 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. 4.

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

More in details, rules (Par_i) for $i \in \{1,2\}$ define an action α_i by $P_1 \mid P_2$ 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 the empty cell S: $S \stackrel{def}{=} w.S' \mathbin{\not+} r.S''$ and $S' \stackrel{def}{=} r.S'$ and $S'' \stackrel{def}{=} r.S'' + e.S''$. The cell S has two potential transitions that can be fired, namely

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

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

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

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

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

where the side-conditions of $(PriSum_2)$ and of (Par_3) are satisfied, blocks at side-condition of rule (Par_1) in red. 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

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

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. 5.

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 rendez-vous 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 (see Def. 3). What we want is that every process using only strong priority satisfies the following *weak local confluence* property [21]. 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[5] 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

[5] 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.

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 $E \stackrel{def}{=} W_1 \,|\, [\cdot] \,|\, W_2$ 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 $W_1 \,|\, S \,|\, W_2 \xrightarrow{\tau}_{\{\,\}} W_1' \,|\, S' \,|\, W_2$ and $W_1 \,|\, S \,|\, W_2 \xrightarrow{\tau}_{\{\,\}} W_1 \,|\, S' \,|\, W_2'$. Still, the two outcomes of the divergence are a critical pair that have no reason to reconverge $W_1' \,|\, S' \,|\, W_2 \to P'$ and $W_1 \,|\, S' \,|\, W_2' \to P'$ 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 $W_1 \,|\, S \xrightarrow{\tau}_{H_1} W_1' \,|\, S'$ and $S \,|\, W_2 \xrightarrow{\tau}_{H_2} S \,|\, W_2'$ have empty blocking 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 $\mathsf{CCS}^{\mathsf{Ph}}$, indeed) 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 27. Note that we cannot simply define $K \stackrel{def}{=} \mathsf{iA}(P) - \{\alpha\}$, because α could be contained in K. For example, take $P \stackrel{def}{=} \alpha.\mathsf{0} \,|\, \alpha.\mathsf{0}$, where $P \xrightarrow[\{\alpha\}]{\alpha}_{\{\alpha\}} \alpha.\mathsf{0}$. 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

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

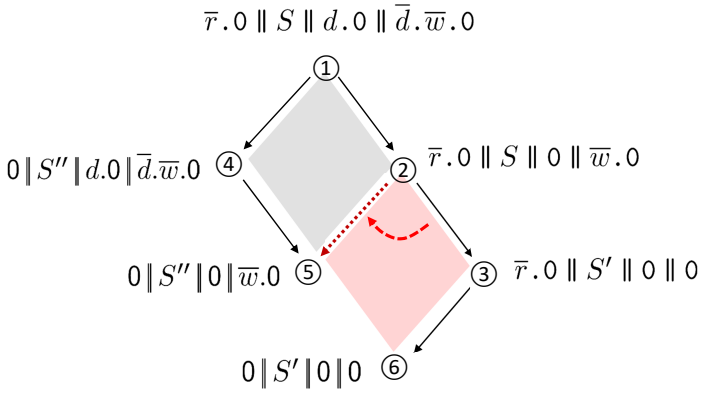


Figure 6 The store S in composition with a reader $\overline{r}.0$ and a delayed writer $d.0 \mid \overline{d}.\overline{w}.0$. The behaviour has two incongruent final outcomes, a store S'' with error in state ⑤ and an error-free store S in state ⑥. The transition system is not confluent. The reason 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.

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 process algebra. 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 $P \dotplus Q$, do 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. 6. 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 [10], described above, are too weak in scheduling control for our purposes and to guaranteeing a weak Church-Rosser property [21]. 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. 6 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 $\overline{d}.0 \,|\, d.\overline{w}.0$ have interacted on action d and advanced together with a silent action τ, i.e. $\overline{d}.0 \,|\, d.\overline{w}.0 \xrightarrow{\tau} 0 \,|\, \overline{w}.0$. 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 $\{r\} \cap \overline{\mathsf{iA}^\tau}(d.0) = \{\,\}$ as well as $\{w, r\} \cap \overline{\mathsf{iA}^\tau}(d.0) = \{\,\}$. 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 $\{w\} \cap \overline{\mathsf{iA}^\tau}(d.0 \,|\, W^*) \neq \{\,\}$. 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. 6 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. 6), 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 $w \in \overline{\mathsf{iA}}^{\tau}(\overline{w}.W')$. 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*[6] and let's call the traditional process algebra with priorities, introduced by [10, 36], as process algebra with *weak priorities*.

As an intermediate step to implement our precedence-based scheduling, the previous example suggests we look at the *weak initial actions* for a process, given in the following definition.

▸ **Definition 28** (Weak Initial Actions & Strong Enabling). *The set $\mathit{iA}^{\tau}(P)$ of weak initial transitions is the smallest extension $\mathit{iA}(P) \subseteq \mathit{iA}^{\tau}(P)$ such that if $\alpha \in \mathit{iA}^{\tau}(Q)$ and $P \xrightarrow{\tau} Q$ then $\alpha \in \mathit{iA}^{\tau}(P)$. Let $\overline{\mathit{iA}}^{\tau}(P) = \overline{\mathit{iA}^{\tau}(P)}$. A transition $P \xrightarrow[R]{\alpha}_H P'$ is called strongly enabled if $H \cap (\overline{\mathit{iA}}^{\tau}(R) \cup \{\tau\}) = \{\,\}$.*

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 $P \,|\, Q \xrightarrow[R]{\tau}_H P' \,|\, Q$. As indicated by the examples of Sec. 3, strong enabling generates confluent reductions in the reflexive case, where rendez-vous actions have a unique sender and a unique receiver process. For multi-cast communication, i.e., non-irreflexive self-blocking c-actions, we need the even stronger notion of *constructive enabling* based on $\overline{\mathsf{iA}}^{*}(R)$ instead of $\overline{\mathsf{iA}}^{\tau}(R)$.

A.2.3 Digression: Strong Priorities lead to 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 $H \cap \overline{\mathsf{iA}}^{\tau}(E) = \{\,\}$, 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

[6] 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.

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[7], 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 $H \cap \overline{\mathsf{iA}}^\tau(E) = \{\}$ 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[8].

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 [5] or StateCharts [17]. 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 $\mathsf{iA}^\tau(E)$. 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) Priorities 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 [18] and PMC [15]. Intuitively, a clock σ is a "broadcast action" in the spirit of Hoare's CSP [19] 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*

[7] 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.

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

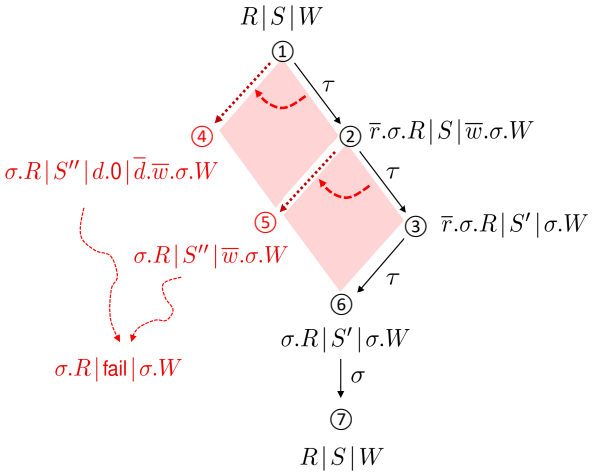


Figure 7 Strong priorities introduce confluence in scheduling.

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 [39] and VHDL [20], or synchronous languages like Esterel [5], Lustre [16], SCCharts [41], BSP [40], 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 $S'' \stackrel{def}{=} w.\mathsf{fail} + r{:}\{w\}.S'' + \sigma{:}\{w,r\}.S$. 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 [18, 11]. 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. 7 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.

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 [29, 30], where processes are *first-class* objects, i.e. sent and received as a values, as in $a!P.Q$ and $a?x.(x \mid Q)$. 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.

Note on Conservativity

The weakly-enabled transitions of unclocked, free processes coincide with the semantics of genuine Milner's CCS [27], as stated in the following easy exercise.

▸ **Exercise 29** (On Def. 2). *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 Milner's original SOS for CCS.*

In fact, one can show that the semantics of weak enabling generates the theory of $\mathsf{CCS}^{\mathsf{Ph}}$ [33]. 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 30** (On Def. 2). *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 restrictions, called *strong enabling* (Def. 28 below) and *constructive enabling* (Def. 3 above) to achieve confluence and secondly we make use of self-loops to control sharing.

Priorities and Scheduling

▸ **Example 31** (On Def. 12). If $A \not\equiv B$ then $a.A + a.B$ is not coherent while $a{:}a.A \mid a{:}a.B$ 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 either. 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. 12 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. 12 as the actions are interfering with each other. ◂

▸ **Example 32** (On Normal form)**.** For the process $P_1 \stackrel{def}{=} (a.c \,|\, b.d \,|\, \overline{a} \,|\, \overline{b}) \setminus \{a,b\}$ we have $(a.c \,|\, d \,|\, \overline{a}) \setminus \{a,b\} \stackrel{\tau}{\leftarrow} P_1 \stackrel{\tau}{\rightarrow} (c \,|\, b.d \,|\, \overline{b}) \setminus \{a,b\}$ 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 rendezvous 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 33** (On Sequentiality à la Esterel in CCS$^{\mathsf{spt}}$)**.** Like CCS, our calculus CCS$^{\mathsf{spt}}$ does not have a generic operator for sequential composition as in 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 [27] (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 $\mathsf{0}$ 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. ◂

The next example illustrates that the closure property for coherence of hiding (Thm. 14(6), see also Prop. 65) needs to require that the hidden clock must not take precedence over rendezvous actions.

▸ **Example 34** (On Clock-hiding)**.** The process $P \stackrel{def}{=} \sigma.A + a{:}\sigma.B \,|\, \sigma$ offers a choice of two transitions $P \xrightarrow[\mathsf{0}]{\sigma}_{\{\,\}} A$ and $P \xrightarrow[\sigma]{a}_{\{\sigma\}} B \,|\, \sigma$. These transitions interfere with each other so that the σ-transition blocks the a-transition under weak enabling, given that $\{\sigma\} \cap \overline{\mathsf{iA}}(\sigma) = \{\sigma\} \neq \{\,\}$. Thus, P is coherent. Note that any policy with $P \Vdash \pi$ must have $\sigma \dashrightarrow a \in \pi$. This does not satisfy the side condition of Thm. 24(6). So we cannot conclude that $P \,/\, \sigma$ is coherent. Let us see what happens when the clock is hidden and thus generates a silent step. Indeed, hiding the clock, we obtain from rule *Hide* the two transitions

$$P \,/\, \sigma \xrightarrow[\mathsf{0} \,/\, \sigma]{\tau}_{\{\,\}} A \,/\, \sigma \text{ and } P \,/\, \sigma \xrightarrow[\sigma \,/\, \sigma]{a}_{\{\}} (B \,|\, \sigma) \,/\, \sigma$$

where the blocking set of the a-transition turns into $\{\sigma\} - \{\sigma\} = \{\,\}$. Now, because σ is removed from the blocking set, the a-transition is even constructively enabled. Moreover, both the τ-transition and the a-transition are interference-free. However, in general, there is no chance that we can find recovergent transitions $A \,/\, \sigma \xrightarrow{a} Q_1$ and $(B \,|\, \sigma) \,/\, \sigma \xrightarrow{\tau} Q_2$ with $Q_1 \equiv Q_2$. Thus, $P \,/\, \sigma$ is not coherent any more. ◂

The problem is this: One the one hand, the *Hide* rule closes the scope of a clock but one the other hand, it does not actually implement any blocking that comes from this clock inside the scope. Such can be implemented by a refined hiding rule that, like (*Com*), implements a "race test" and introduces a τ into the blocking set whenever there is a conflict:

$$\frac{P \xrightarrow[R]{\alpha}_H Q \quad H' = H - \{\sigma\} \cup \{\tau \mid \sigma \in \overline{\mathsf{iA}}^*(R) \cap H\}}{P \,/\, \sigma \xrightarrow[R \,/\, \sigma]{\alpha \,/\, \sigma}_{H'} Q \,/\, \sigma} \; (\mathit{Hide}^*)$$

In this manner, the enabling check for the conclusion transition verifies that $\tau \notin H'$ and thus that $\sigma \notin \overline{\mathsf{iA}}^*(R) \cap H$. In other words, if the local clock σ is in H (blocking the visible action α) then the concurrent context R inside the clock scope must not conflict with σ. Not surprisingly, *mutatis mutandis*, a similar refinement could be made to the restriction rule giving

$$\frac{P \xrightarrow[R]{\alpha}_H Q \quad \alpha \notin \{\ell, \overline{\ell}\} \quad H' = H - \{\ell, \overline{\ell}\} \cup \{\tau \mid \ell \in \overline{\mathsf{iA}}^*(R) \cap H\}}{P \setminus \ell \xrightarrow[R \setminus \ell]{\alpha}_{H'} Q \setminus \ell} \; (\mathit{Restr}^*)$$

In this paper we stick to the simpler versions (*Hide*) and (*Restr*) but conjecture that the side conditions on the policy in Thm. 24(6,7) can be lifted to show that for (*Hide**) and (*Restr**) the operations of hiding and restriction preserve coherence.

▸ **Example 35** (Clock-hiding with built-in Scheduling Test)**.** Consider the process $P \stackrel{def}{=} \sigma.A + a{:}\sigma.B \,|\, b.\sigma$ where the local clock σ also blocks the action a by priority. But now the clock is guarded in $b.\sigma$ and so to become active for P, it needs the synchronisation with an external action $\overline{b}$. As in the previous example $P \,/\, \sigma$ would not be coherent. The extended hiding rule (*Hide**) would generate the a-transition

$$P \,/\, \sigma \xrightarrow[(b.\sigma)\,/\,\sigma]{a}\!\!\!\!\rightarrow_{\{\tau\}} (B \,|\, b.\sigma) \,/\, \sigma \text{ from } P \xrightarrow[b.\sigma]{a}\!\!\!\!\rightarrow_{\{\sigma\}} A \,|\, b.\sigma.$$

The τ enters into the blocking set because of the race test which discovers that $\sigma \in \overline{\mathsf{iA}}^*(b.\sigma) \cap \{\sigma\}$, where $\{\sigma\}$ is the blocking set of the a-transition in P. Hence, the a-transition of $P \,/\, \sigma$ is not (constructively) enabled anymore, which saves coherence. There is no requirement to exhibit any reconvergence for it. ◂

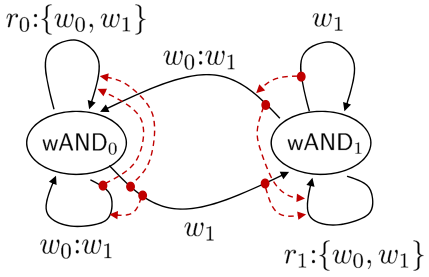
Policies

▸ **Example 36** (On coherence and pivotable processes)**.** 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 rendezvous synchronisation $\tau = a \,|\, \overline{a}$ is blocked by the presence of the offering $\overline{b}$ from a *third* thread. We have $P_1 \xrightarrow[\overline{b}]{\tau}\!\!\!\!\rightarrow_{\{b\}} \overline{b}$, which is not weakly enabled, because $\{b\} \cap \overline{\mathsf{iA}}(\overline{b}) \neq \{\,\}$. The blocking is due to the concurrent context $\overline{b}$ 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 race test of rule (*Com*) which introduces τ into the blocking set $P_2 \xrightarrow[0]{\tau}\!\!\!\!\rightarrow_{\{b,\tau\}} 0$ because $\{b\} \cap \overline{\mathsf{iA}}(a + \overline{b}) \not\subseteq \{\overline{a}\}$. This transition therefore is not weakly enabled, because of $\{b, \tau\} \cap (\overline{\mathsf{iA}}(0) \cup \{\tau\}) \neq \{\,\}$. 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 $P_3 \xrightarrow[0]{\tau}\!\!\!\!\rightarrow_{\{b\}} \overline{b}$. 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 race condition of (*Com*). ◂

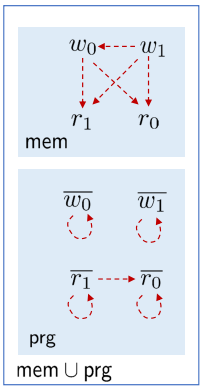
▸ **Example 37** (On Coherent and Pivotable Processes)**.** The process $P_1 \stackrel{def}{=} \sigma + a{:}\sigma \,|\, \overline{a}$ is coherent and pivotable, with policy $\sigma \dashrightarrow a \in \pi$ and $\sigma \diamond \overline{a} \in \pi$. It has both $a, \overline{a} \in \mathsf{iA}(P)$ 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 precedence relationship with σ, P_2 does not satisfy concurrent independence of σ with neither a nor $\overline{a}$. Formally, any policy π for P_2 must have $\sigma \dashrightarrow a \in \pi$ and $\overline{a} \dashrightarrow \sigma \in \pi$ violating the pivot requirement that if $\sigma \dashrightarrow a \in \pi$, then $\sigma \diamond \overline{a} \in \pi$. 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 $\tau = \alpha \,|\, \overline{\alpha}$ that has σ in its blocking set. ◂

An interesting special class of pivot policies are input-scheduled policies.

▸ **Definition 38** (Input-scheduled Processes)**.** *A policy π is called input-scheduled if $\pi \subseteq \pi_{is}$ where $\pi_{is} = (L_{is}, \dashrightarrow_{is})$ given by alphabet $L_{is} = \mathcal{L}$ and precedence relation $\ell_1 \dashrightarrow \ell_2 \in \pi_{is}$ iff $\ell_1 = \ell_2$ or $\ell_1, \ell_2 \in \mathcal{A} \cup \mathcal{C}$. A process is input-scheduled if P conforms to an input-scheduled policy.*



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



(b) The mem ∪ prg Pivot Policy.

Figure 8 A boolean memory process.

▸ **Example 39** (On Pivotable but not Input-scheduled Processes)**.** The process $a + b{:}c \mid \overline{a} + \overline{b}{:}\overline{a}$ is pivotable. It conforms to π with precedences $c \dashrightarrow b \in \pi$ and $\overline{a} \dashrightarrow \overline{b} \in \pi$, while at the same time $\overline{c} \diamond \overline{b} \in \pi$ and $a \diamond b \in \pi$. However, assuming $a \neq b$, the process is not input-scheduled, because it has $\overline{a} \dashrightarrow \overline{b} \in \pi$ while $\overline{a} \diamond \overline{b} \in \pi_{is}$. The same is true of the process in Ex. 5. ◂

▸ **Example 40** (On Read/Write Process, reloaded)**.** Fig. 8 show that the processes wAND$_v$ conform to a policy mem with $w_i \dashrightarrow r_j \in$ mem for $i \neq j$ and $w_1 \dashrightarrow w_0 \in$ mem. 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 $Q \xrightarrow{\ell}_H Q'$ and $\ell \notin H$ then there is a repetition $Q' \xrightarrow{\ell}_{H'} Q''$ with $H' = H$. This yields Def. 12. Secondly, suppose $Q \xrightarrow{\ell_1}_{H_1} Q_1$ and $Q \xrightarrow{\ell_2}_{H_2} Q_2$. 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 $Q_1 \not\equiv Q_2$ then necessarily $\ell_1 \neq \ell_2$. Thus, Def. 12 is satisfied, too, for trivial reasons. So, the memory, which satisfies wAND$_v \Vdash$ mem, is pivotable.

Note that wAND$_v$ is not a reflexive 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 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 that is coherent and conformant to the policy prg with $\overline{r_1} \dashrightarrow \overline{r_i} \in$ prg and $\overline{r_0} \dashrightarrow \overline{r_0} \in$ prg. Now we observe that the union mem ∪ prg of policies is a pivot policy. Since wAND$_v \Vdash$ mem and $P \Vdash$ prg we have wAND$_v \Vdash$ mem ∪ prg as well as $P \Vdash$ mem ∪ prg. Then, by Prop. 64 we get

$$\text{wAND}_v \mid P \Vdash \text{mem} \cup \text{prg}$$

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 wAND$_v \mid P_1 \mid P_2 \mid \cdots \mid P_n$ will be coherent for mem ∪ prg. In particular, all reductions will be

confluent under strong enabling. Technically, what happens is that first all writers of value 1, synchronising as $w_1 \,|\, \overline{w_1}$, will go ahead. Then all writers of $\mathsf{0}$, synchronising as $w_0 \,|\, \overline{w_0}$ become enabled. Only then the readers will be able to retrieve the final value via $r_0 \,|\, \overline{r_0}$ or $r_1 \,|\, \overline{r_1}$ rendezvous synchronisations. ◂

Observe that the pivot policy $\mathsf{mem} \cup \mathsf{prg}$ of Ex. 40 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 an input-scheduled process in the sense of Def. 38. This breaks the assumption on which the theory of classical prioritised process algebras like [10] or [36] 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. [33]*

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 $\mathsf{CCS}^{\mathsf{CW}}$.

Note that all the examples of Sec. 3 have the "input-scheduled" property that (non-trivial, i.e. non-self-blocking) precedences only exist between receiver actions $\mathcal{A}$, while sender actions $\overline{\mathcal{A}}$ never appear in any choice contexts. These are the processes covered by [10] which restrict prioritised choice to receiver actions ("input-guarded, prioritised choice") in the same way as the programming language Occam [4] with its "PRIALT" construct does (cf. [10]).

▸ **Example 41.** The memory cell schedules readers after the writers and among the writers prefers to engage first with with those that set the value to $\mathsf{0}$, 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 boolean 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 $\overline{w_1}{:}\overline{w_1} \,|\, \overline{w_0}{:}\overline{w_0} \,|\, \mathsf{wAND}_0 \,|\, R$ and $\overline{w_0}{:}\overline{w_0} \,|\, \overline{w_1}{:}\overline{w_1} \,|\, \mathsf{wAND}_0 \,|\, R$ 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 $\overline{w_1}{:}\overline{w_1}.\overline{w_0}{:}\overline{w_0} \,|\, \mathsf{wAND}_0 \,|\, R$ and $\overline{w_0}{:}\overline{w_0}.\overline{w_1}{:}\overline{w_1} \,|\, \mathsf{wAND}_0 \,|\, R$ 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. ◂

Strong Enabling vs Constructive Enabling

As an intermediate step to implement our precedence-based scheduling, the previous example suggests we look at the *weak initial actions* for a process. The next example shows why strong enabling, based on $\overline{\mathsf{iA}}^{\tau}(R)$, is not sufficient and why we need the stronger notion of c-enabling (Def. 3), based on $\overline{\mathsf{iA}}^{*}(R)$.

▸ **Example 42** (On Multi-Cast and Constructive Enabling). Suppose $S_0 \stackrel{def}{=} a{:}e.S_0 + e.S_1 + \sigma{:}\{a,e\}.S_0$ is an Esterel Signal (Sec. 9) in its absent state, with abbreviations $e = emit$, $a = abs$ and $p = pres$. 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 $R \stackrel{def}{=} \overline{p}{:}\overline{p}.R_1 + \overline{a}{:}\{\overline{a},\overline{p}\}.R_0$. 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 $\overline{p}$, $\overline{a}$ (readers) but at most one receiver for p, a (signal).

Consider a reader $C \stackrel{def}{=} \overline{a}{:}\overline{a}.\overline{e}{:}\overline{e}.C_1$ 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 $D \stackrel{def}{=} \overline{a}{:}\overline{a}.\overline{e}{:}\overline{e}.D_1$ 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 rendezvous 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 $S_0 \,|\, \overline{a}{:}\overline{a} \xrightarrow[S_0\,|\,\mathsf{0}]{\tau}_{\{e,a\}} S_0 \,|\, \mathsf{0}$ would block, because $\overline{a} \in \overline{\mathsf{iA}}(S_0) \subseteq \overline{\mathsf{iA}}^{\tau}(S_0 \,|\, \mathsf{0})$, 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. 3) that closes under *all* non-clock actions of E. In this case, we do find $e \in \{e,\overline{a}\} \cap \overline{\mathsf{iA}}^{*}(\mathsf{0}\,|\,\mathsf{0}\,|\,D) \neq \{\,\}$. In this way, diverging reductions are no longer enabled. On the other hand, the reduction $S_0 \,|\, C \xrightarrow[\mathsf{0}\,|\,\mathsf{0}]{\tau}_{\{e,a\}} S_0 \,|\, \overline{e}{:}\overline{e}.C_1$

$$
\begin{aligned}
\mathsf{ABRO} &\stackrel{def}{=} \sigma.(\mathsf{A} \mid \mathsf{B} \mid \mathsf{R} \mid \mathsf{O}) \setminus \{s,t\} \\
\mathsf{A} &\stackrel{def}{=} k{:}k.\mathsf{0} + a{:}\{k,a\}.\overline{s}{:}\overline{s}.\mathsf{0} + \sigma{:}\{k,a\}.\mathsf{A} \\
\mathsf{B} &\stackrel{def}{=} k{:}k.\mathsf{0} + b{:}\{k,b\}.\overline{s}{:}\overline{s}.\mathsf{0} + \sigma{:}\{k,b\}.\mathsf{B} \\
\mathsf{R} &\stackrel{def}{=} r{:}r.\mathsf{R}' + \tau{:}r.\sigma.\mathsf{R} \\
\mathsf{R}' &\stackrel{def}{=} \overline{k}.\mathsf{R}' + \tau{:}\overline{k}.\mathsf{ABRO} \\
\mathsf{O} &\stackrel{def}{=} k{:}k.\mathsf{0} + s{:}k.\mathsf{O} + t{:}\{k,t,s\}.\overline{o}{:}\overline{o}.\mathsf{0} + \sigma{:}\{k,s,t\}.\mathsf{O}.
\end{aligned}
$$

Figure 9 Simplified version of ABRO.

remains enabled, because $\overline{\mathsf{iA}}^*(\mathsf{0} \mid \mathsf{0}) = \{\}$. In general, single readers $S_0 \mid C$ where C is an arbitrary sequential process, or multiple readers $S_0 \mid C \mid D$ where *at most one* of C or D is writing, i.e., sending $\overline{e}$, remain enabled. ◂

▸ **Example 43** (On Fully Multi-Cast ABRO)**.** To model full multi-cast ABRO in 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. Consider the process ABRO from Ex. 10 again:

- 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 $r{:}r.\overline{k}_1{:}\overline{k}_1.\overline{k}_2{:}\overline{k}_2.\overline{k}_3{:}\overline{k}_3.\overline{k}_4{:}\overline{k}_4.\mathsf{ABRO}$ 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 $R \xrightarrow[\mathsf{0}]{\tau}_{\{r\}} \sigma.\mathsf{R}$ 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 $\overline{\mathsf{iA}}^*(\sigma.R) = \{\}$, 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 0 where it is inactive but permits 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 $\mathsf{A} \mid \overline{k}_1.\mathsf{0} \mid \overline{a}.\mathsf{0} \xrightarrow[\mathsf{0} \mid \mathsf{0} \mid \overline{a}.\mathsf{1}]{\tau}_{\{k_1\}} \mathsf{0} \mid \mathsf{0} \mid \overline{a}.\mathsf{0}$ is constructively enabled while $\mathsf{A} \mid \overline{k}_1.\mathsf{0} \mid \overline{a}.\mathsf{0} \xrightarrow[\mathsf{0} \mid \overline{k}_1.\mathsf{0} \mid \mathsf{0}]{\tau}_{\{k_1,a\}} \mathsf{0} \mid \overline{k}_1.\mathsf{0} \mid \mathsf{0}$ 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 $\tau = \overline{t}{:}\{k_3, \overline{t}, s\} \,|\, t{:}\{k_4, t\}$ 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 $s \in \overline{\mathsf{iA}}^*(\mathsf{A} \,|\, \mathsf{B})$. Only when both prefix sequences $a{:}\{a, k_1\}.\overline{s}{:}\overline{s}.\mathsf{0}$ and $b{:}\{b, k_2\}.\overline{s}{:}\overline{s}.\mathsf{0}$ have been eliminated by reductions, then the termination can occur.

Using multi-cast, we can improve on this as seen in Fig. 9: 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 (multi-cast) 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 that point, it takes a clock step to repeat ABRO from the start.

One can verify that ABRO is coherent and pivotable using Thm. 24.

- The choices in each of the threads A, B, R, O are fully resolved by the blocking sets, i.e., every pair of actions in the summations are distinct and in precedence relationships. Every rendezvous 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 A, B, R, O themselves are coherent.
- A suitable policy π, for which the threads A, B, R, O are coherent, enforces the precedences
  - $k \dashrightarrow \ell \in \pi$ for $\ell \in \{k, a, b, s, t, \sigma\}$
  - $\ell \dashrightarrow \ell \in \pi$ for $\ell \in \{r, k, a, b, \overline{s}, t, \overline{o}\}$
  - $s \dashrightarrow t \in \pi$
  - $\ell \dashrightarrow \sigma \in \pi$ 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, $\overline{k} \diamond \overline{\ell} \in \pi$ for all $\ell \in \{r, o, a, b, s, t, \sigma\}$, $\overline{s} \diamond \overline{t} \in \pi$ and $\overline{\ell} \diamond \sigma \in \pi$ for $\ell \in \{r, o, a, b, s, t\}$.
- Thus, the policy π (and thus $\pi \setminus \mathcal{R}$) is a pivot policy and so the parallel composition $\mathsf{A} \,|\, \mathsf{B} \,|\, \mathsf{R} \,|\, \mathsf{O}$ is coherent for π. Further, π is precedence-closed for $\{s, t\}$ which means that $(\mathsf{A} \,|\, \mathsf{B} \,|\, \mathsf{R} \,|\, \mathsf{O}) \setminus \{s, t\}$ is coherent for $\pi \setminus \{s, t\}$. Finally, since $\{\} \subseteq \{\ell \mid \ell \dashrightarrow \sigma \in \pi \setminus \{s, t\}\} = \{a, b\}$, the process ABRO (which starts with a prefix $\sigma{:}H$ where $H = \{\,\}$) is coherent for $\pi \setminus \{s, t\}$. ◂

C Auxiliary Results

In the following we list some simple observation on the properties of $\mathsf{iA}(P)$, $\mathsf{iA}^\tau(P)$ and $\mathsf{iA}^*(P)$ that will be used in the proofs.

▸ **Lemma 44.** *Let P, Q be processes:*

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

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

(3) *$\mathit{iA}(P) \cap \mathcal{R} \subseteq \mathit{iA}(P \,|\, Q)$.*

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

(5) *If $P \xrightarrow[R]{\alpha}_H Q$ then $\mathit{iA}(R) \subseteq \mathit{iA}(P)$.*

Proof. (2) is by definition. The other points follow by an inspection of the SOS rules, using the definition of $\mathsf{iA}(P)$. ◂

Lem. 44(1) means that clock actions always run in an empty concurrent environment. Observe that Lem. 44(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 (*Com*) 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. 44(4) implies that single-threaded processes are never blocked.

▸ **Lemma 45.** *Let P be an arbitrary process:*
(1) $\mathit{iA}(P) \cap \mathcal{L} \subseteq \mathit{iA}^{\tau}(P) \subseteq \mathit{iA}^{*}(P)$.
(2) *If $P \equiv Q$ then $\mathit{iA}^{*}(P) = \mathit{iA}^{*}(Q)$.*
(3) *If $P \xrightarrow{\alpha} Q$ for $\alpha \in \mathcal{R} \cup \{\tau\}$, then $\mathit{iA}^{*}(Q) \subseteq \mathit{iA}^{*}(P)$.*

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

On Interference

▸ **Proposition 46.** *Suppose $\alpha_1{:}H_1[E_1 \,|\, R]$ and $\alpha_2{:}H_2[E_2 \,|\, R]$ 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 $H_2' \subseteq H_2$.*

As a special case, choosing $R = 0$, Proposition 46 implies non-interference of $\alpha_i{:}H_i'[E_i]$ for $i \in \{1,2\}$ from non-interference of $\alpha_i{:}H_i[E_i]$ for any $H_i' \subseteq H_i$.

Proof. 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. 44, $\overline{\mathsf{iA}}^{*}(E_i) \subseteq \overline{\mathsf{iA}}^{*}(E_i \,|\, R)$, the inclusions $H_i' \subseteq H_i$ preserve this property, i.e., $\tau \notin H_i'$ and $H_i' \cap \overline{\mathsf{iA}}^{*}(E_i) \subseteq H_i \cap \overline{\mathsf{iA}}^{*}(E_i \,|\, R) = \{\,\}$. ◂

On Policies

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. By way of this ordering, we can 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. The $\preceq$-minimal policy π_{min} has empty alphabet and empty precedences. It is self-dual, i.e., $\overline{\pi}_{min} = \pi_{min}$.

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.

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 $\overline{\ell}_1 \dashrightarrow \overline{\ell}_2 \in \overline{\pi}$ iff $\ell_1 = \ell_2$ or $\ell_1 \diamond \ell_2 \in \pi$, i.e., both $\ell_1 \dashrightarrow \ell_2 \notin \pi$ and $\ell_2 \dashrightarrow \ell_1 \notin \pi$. 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 [12]. Analogously, $\overline{\overline{\pi}}$ expresses the dependencies of actions in π, called the associated *dependence alphabet*.

▸ **Example 47.** Suppose $P \Vdash \ell_1 \diamond \ell_2$ then all transitions with labels ℓ_1 and ℓ_2 of P must be from concurrent threads and thus cannot block each other. For instance, both $\ell_1 \,|\, \ell_2$ and $\ell_1 + \ell_2$ conform to $\ell_1 \diamond \ell_2$. However, the choice $\ell_1 + \ell_2$ is not coherent. It becomes coherent if we add a precedence $\ell_1 + \ell_2{:}\ell_1$ but then it is not longer conformant to $\ell_1 \diamond \ell_2$. Instead, we now have $\ell_1 + \ell_2{:}\ell_1 \Vdash \ell_1 \dashrightarrow \ell_2$, where the policy expresses the precedence coded in the blocking sets of the prefixes. Suppose $P \Vdash \ell_1 \diamond \ell_2$ then all transitions with labels ℓ_1 and ℓ_2 of P must be from concurrent threads and thus cannot block each other. For instance, both $\ell_1 \,|\, \ell_2$ and $\ell_1 + \ell_2$ conform to $\ell_1 \diamond \ell_2$. However, the choice $\ell_1 + \ell_2$ is not coherent. It becomes coherent if we add a precedence $\ell_1 + \ell_2{:}\ell_1$ but then it is no longer conformant to $\ell_1 \diamond \ell_2$. Instead, we now have $\ell_1 + \ell_2{:}\ell_1 \Vdash \ell_1 \dashrightarrow \ell_2$, where the policy expresses the precedence coded in the blocking sets of the prefixes. More generally, if $\ell_1 \dashrightarrow \ell_2 \in \pi$ and $\ell_1 \neq \ell_2$ are distinct, then $P \Vdash \pi$ may take ℓ_1 and ℓ_2 transitions to completely different states from the same thread.

▸ **Definition 48** (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 49.** *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 $\ell_1 \diamond \ell_2 \in \pi$ or $\overline{\ell}_1 \diamond \overline{\ell}_2 \in \pi$.*

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 $\ell_1 \diamond \ell_2 \notin \pi$. Hence $\ell_1 \dashrightarrow \ell_2 \in \pi$ or $\ell_2 \dashrightarrow \ell_2 \in \pi$. Consider the first case, i.e., $\ell_1 \dashrightarrow \ell_2 \in \pi$. 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 $\overline{\overline{\ell}}_1 \dashrightarrow \overline{\overline{\ell}}_2 \in \overline{\pi}$. Now, the definition of the dual policy means that $\overline{\ell}_1 \dashrightarrow \overline{\ell}_2 \notin \pi$ and $\overline{\ell}_2 \dashrightarrow \overline{\ell}_1 \notin \pi$. This is the same as $\overline{\ell}_1 \diamond \overline{\ell}_2 \in \pi$ 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 $\overline{\ell}_1 \dashrightarrow \overline{\ell}_2 \in \pi$ which implies $\overline{\ell}_1 \diamond \overline{\ell}_2 \notin \pi$. If $\ell_1 = \ell_2$ we have $\overline{\ell}_1 \dashrightarrow \overline{\ell}_2 \in \overline{\pi}$ directly by definition. So, let $\ell_1 \neq \ell_2$. Because of $\overline{L} \subseteq L$ and (2) this gives us $\ell_1 \diamond \ell_2 \in \pi$ considering again that $\ell_i = \overline{\overline{\ell}}_i$. But $\ell_1 \diamond \ell_2 \in \pi$ is the same as $\overline{\ell}_1 \dashrightarrow \overline{\ell}_2 \in \overline{\pi}$ by definition. ◂

▸ **Proposition 50.** *Suppose P is input-scheduled π_{is} 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}. ◂

The next crucial Lemma implies that two coherent threads cannot block each other if they are conformant to the same pivot policy. The blocking of a reduction $\ell \,|\, \overline{\ell}$ must always arise from a thread in the concurrent context that is not involved in the synchronisation.

▸ **Lemma 51.** *Let $P_1 \Vdash \pi$ and $P_2 \Vdash \pi$ 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, $H_2 \cap \overline{\mathit{iA}}(R_1) = \{\,\}$ implies $H_2 \cap \overline{\mathit{iA}}(P_1) \subseteq \{\overline{\ell}\}$.*

Proof. Let $\ell \in \mathcal{L}$ and the transitions be given as in the statement of the Lemma. Suppose $\alpha \in H_2$ and $\alpha \in \overline{\mathsf{iA}}(P_1)$. We claim that $\alpha = \overline{\ell}$, whence $H_2 \cap \overline{\mathsf{iA}}(P_1) \subseteq \{\overline{\ell}\}$. By contraposition let us assume that $\alpha \neq \overline{\ell}$, or equivalently $\overline{\alpha} \neq \ell$. Further, since $\mathsf{iA}(P_1) \subseteq \mathcal{L}$ we also have $\alpha \neq \tau$. The assumption $\alpha \in H_2$ means $\alpha \dashrightarrow \overline{\ell} \in \pi$ by Def. 12(1) applied to P_2. Since π is pivot we must thus have $\overline{\alpha} \diamond \ell \in \pi$, specifically $\overline{\alpha} \dashrightarrow \ell \notin \pi$, i.e., $\overline{\alpha} \notin H_1$, again by Def. 12(1), now applied to P_1. But since $\overline{\alpha} \in \mathsf{iA}(P_1)$, by Lem. 44, there is a transition $P_1 \xrightarrow[E]{\overline{\alpha}}_H Q$. Since also $\ell \dashrightarrow \overline{\alpha} \notin \pi$, by pivot property, it follows $\ell \notin H$ by Def. 12(1) again for P_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 (Def. 53). Since $\ell \neq \overline{\alpha}$ there must exist a reconvergent process R and transitions

$$P_1' \xrightarrow[E']{\overline{\alpha}}_{H'} R \text{ and } Q \xrightarrow[R_1']{\ell}_{H_1'} R \text{ with } R_1 \xrightarrow{\overline{\alpha}} R_1' \text{ and } E \xrightarrow{\ell} E'.$$

The latter transitions are strengthenings of the residuals $R_1 \overset{\overline{\alpha}}{\rightsquigarrow} R_1'$ and $E \overset{\ell}{\rightsquigarrow} E'$ that must exist according to the statement of Coherence, because of $\{\ell, \overline{\alpha}\} \subseteq \mathcal{R}$ and $\ell \neq \overline{\alpha}$. However, the transition $R_1 \xrightarrow{\overline{\alpha}} R_1'$ implies $\overline{\alpha} \in \mathsf{iA}(R_1)$ which is impossible, because then $\alpha \in H_2 \cap \overline{\mathsf{iA}}(R_1)$ while $H_2 \cap \overline{\mathsf{iA}}(R_1) = \{\,\}$ by assumption.

◂

We can show that pivotable processes do not have internal races.

▸ **Lemma 52.** *Let $P \Vdash \pi$ 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 (*Com*), i.e.,

$$\frac{P_1 \xrightarrow[R_1]{\ell}_{H_1} P_1' \quad P_2 \xrightarrow[R_2]{\overline{\ell}}_{H_2} P_2' \quad B = \{\ell \,|\, \overline{\ell} \mid H_2 \cap \overline{\mathsf{iA}}(P_1) \nsubseteq \{\overline{\ell}\} \text{ or } H_1 \cap \overline{\mathsf{iA}}(P_2) \nsubseteq \{\ell\}\}}{P_1 \,|\, P_2 \xrightarrow[R_1 \,|\, R_2]{\ell \,|\, \overline{\ell}}_{H_1 \cup H_2 \cup B} P_1' \,|\, P_2'} \; (\mathit{Com})$$

for some $\ell \in \mathcal{L}$. The assumption $H \cap \overline{\mathsf{iA}}(E) = \{\,\}$ for the top-level reduction implies that $(H_1 \cup H_2) \cap \overline{\mathsf{iA}}(R_1 \,|\, R_2) = \{\,\}$ at the point of (*Com*). Here we may assume, by induction on the depth of the derivation, that $\tau \notin H_1 \cup H_2$. We must show that the synchronisation action τ cannot enter into the blocking set by (*Com*), 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 $P_1 \Vdash \pi'$ and $P_2 \Vdash \pi'$ are also coherent for a common pivot policy π'. This policy may contain local labels that are restricted or hidden in the outer process P. Since by assumption we have $H_1 \cap \overline{\mathsf{iA}}(R_1) \subseteq (H_1 \cup H_2) \cap \overline{\mathsf{iA}}(R_1 \,|\, R_2) = \{\,\}$ and likewise $H_2 \cap \overline{\mathsf{iA}}(R_2) = \{\,\}$, we can apply Lem. 51 to show $H_2 \cap \overline{\mathsf{iA}}(P_1) \subseteq \{\overline{\ell}\}$ and $H_1 \cap \overline{\mathsf{iA}}(P_2) \subseteq \{\ell\}$. This means $B = \{\,\}$ and we are done. ◂

The Role of Pivotability

Let us give an example to show that the requirement of pivotability cannot be removed in the closure statement for parallel compositions of Thm. 14. For compactness of notation let us write $\underline{\ell}{:}H$ for $\ell{:}(H \cup \{\ell\})$ to create reflexive prefixes. Let

$$Q = \underline{b} \,|\, \underline{c}{:}a \text{ and } R = \overline{\underline{b}}{:}\overline{a}.R_1 + \overline{\underline{c}}{:}\overline{a}.R_2 \text{ with } R_1 = \overline{\underline{a}} + \overline{\underline{c}}{:}\overline{a} \text{ and } R_2 = \overline{\underline{a}} + \overline{\underline{b}}{:}\overline{a}.$$

First, we observe that both Q and R are coherent:

- Process Q has two admissible transitions

$$Q \xrightarrow[0]{b}_{\{b\}} 0 \text{ and } Q \xrightarrow[0]{c}_{\{c,a\}} 0$$

which both are self-interfering. Thus, reconvergence is required only between the two transitions which is guaranteed, because they are from concurrent threads. This can also be obtained from Thm. 24 and the pivot policy with $b \dashrightarrow b \in \pi$, $c \dashrightarrow c \in \pi$ and $a \dashrightarrow c \in \pi$.

- Process R has two admissible transitions, the c-action $R \xrightarrow[0]{\overline{b}}_{\{\overline{a}\}} R_1$ and $R \xrightarrow[0]{\overline{c}}_{\{\overline{a}\}} R_2$. Each interferes with itself but not with the other. Thus, coherence implies that R_1 and R_2 reconverge. This is indeed the case, since we have $R_1 \xrightarrow[0]{\overline{c}}_{\{\overline{a}\}} 0$ and $R_2 \xrightarrow[0]{\overline{b}}_{\{\overline{a}\}} 0$, with commuting actions. The concurrent contexts are residual since $0 \overset{\overline{c}}{\rightsquigarrow} 0$ and $0 \overset{\overline{b}}{\rightsquigarrow} 0$. Since all admissible transitions of R_1 (and R_2) are reflexive and interfere with each other, R_1 (and R_2) are coherent, too.

Next we show that the parallel composition $Q \mid R$ is not coherent. We have admissible c-actions

$$Q \mid R \xrightarrow[\underline{c}:a]{b \mid \overline{b}}_{\{b,\overline{b},\overline{a}\}} \underline{c}:a \mid R_1 \text{ and } Q \mid R \xrightarrow[\underline{b}]{c \mid \overline{c}}_{\{c,\overline{c},a,\overline{a}\}} \underline{b} \mid R_2.$$

which are in fact c-enabled. However, the continuation process $\underline{c}:a \mid R_1 = \underline{c}:a \mid \overline{a} + \overline{c}:\overline{a}$ on the left is binary blocked by a race condition. We find

$$\underline{c}:a \mid R_1 \xrightarrow[0]{c \mid \overline{c}}_{\{c,\overline{c},\overline{a},\tau\}} 0$$

where the silent τ enters the blocking set, because $\{a\} \cap \overline{\mathsf{iA}}(\overline{a} + \overline{c}:\overline{a}) \not\subseteq \{c\}$. This is a race computed in rule (*Com*). Thus, the reduction $c \mid \overline{c}$ of the original state $Q \mid R$ is not longer c-enabled in the state $\underline{c}:a \mid R_1$. This breaks coherence.

Finally, we notice that $Q \mid R$ is not pivotable. A common policy π for $Q \mid R$ must obviously include the precedences $\overline{a} \dashrightarrow \overline{c} \in \pi$, as well as $a \dashrightarrow c \in \pi$. This is not pivot according to Def. 48.

On Milner' Confluence Class

As we have noted above, CCS corresponds to the unclocked and (blocking) free processes of $\mathsf{CCS}^{\mathsf{spt}}$ (Prop. 29). For these processes, admissible transitions and strongly enabled transitions coincide. The key result of Milner [27] (Chap. 11, Prop. 19) is that confluence is preserved by inaction 0, prefixing $\ell.P$ and *confluent composition* defined as $P \mid_L Q = (P \mid Q) \setminus L$ 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, $\ell{:}\ell \mid \overline{\ell}{:}\overline{\ell} \mid \overline{\ell}{:}\overline{\ell}$ 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 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, $\ell_1 \dashrightarrow \ell_2 \in \pi_P$ 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. 48) and dependency-closed for all label sets (Def. 48), 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. 60), self-blocking action prefix $\ell{:}\{\ell\}.Q$ (Prop. 61), parallel composition $P \,|\, Q$ (Prop. 64) and restriction $P \setminus L$ (Prop. 66). 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. 12(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 $Q_1 \not\equiv Q_2$ 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 (Com) 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.

D Proofs

First, we adopt a slightly more general form of interference, for general c-actions not just for those that are constructively enabled, as in Def. 11.

▸ **Definition 53.** *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 $H_i \cap (\overline{\mathit{iA}}^*(E_i) \cup \{\tau\}) = \{\,\}$.*

The extra condition (2) in Def. 53 says that for a c-action not to interfere with a silent (and thus uncontrollable) action it must be c-enabled. Obviously, Def. 53 is a weakening of Def. 11, i.e. if $\alpha_1{:}H_1[E_1]$ and $\alpha_2{:}H_2[E_2]$ are interference-free according to Def. 53 they are interference-free according to Def. 11.

▸ **Proposition 54.** *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 $\sigma \in H_1$.*

Proof. Since $R_2 = \mathsf{0}$ by Lem. 44(1) and $\mathsf{iA}(\mathsf{0}) = \{\,\}$, we have $\alpha \notin \mathsf{iA}(R_2)$ and thus α cannot be an initial action $R_2 \xrightarrow{\alpha} R_2'$, by Lem. 44(2). Now we have $\{\alpha, \sigma\} \cap \mathcal{C} \neq \{\,\}$. But then the coherence Def. 12(2) implies that the c-actions $\alpha{:}H_1[R_1]$ and $\sigma{:}H_2[R_2]$ must interfere. Since $\alpha \neq \sigma$ this implies $\alpha \in H_2$ or $\sigma \in H_1$, or $\alpha = \tau$ and $H_2 \cap (\overline{\mathsf{iA}}^*(R_2) \cup \{\tau\}) \neq \{\,\}$. Since $R_2 = \mathsf{0}$ the latter reduces to $\alpha = \tau \in H_2$. The last part of the Proposition now follows from the fact that the clock transition is enabled and thus $\tau \notin H_2$. ◂

Of Thm. 14. 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. 12(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. 45 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. 15. The first part of the statement follows directly from Def. 12(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. 44(1) we have $R_i = 0$ and thus $\mathsf{iA}(R_i) = \{\}$ when $\ell \in C$. ◂

▸ **Lemma 55.** *Let $P_1 \Vdash \pi$ and $P_2 \Vdash \pi$ be coherent for the same pivot policy π. If P_1 and P_2 are single-threaded, then every admissible transition of $P_1 \mid P_2$ is c-enabled.*

Proof. Consider an admissible transition of $P_1 \mid P_2$ where P_1 and P_2 are single-threaded. Firstly, by Lem. 44, 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 $\tau{:}(H_1 \cup H_2 \cup B)[R_1 \mid R_2]$ with $R_1 \mid R_2 = 0$. Because of Lem. 51, 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 $\ell_1 \mid \ell_2$ is c-enabled, too. ◂

The following, slightly extended version of Prop. 23 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 56** (Sequential Schedule and 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 $P \xrightarrow{\tau} P'$.*

Proof. The proof proceeds by contradiction. Let $P \Vdash \pi$ 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. 54 we infer $\sigma \in N$ or $\ell \in H$, i.e., by conformance Def. 17 we have $\pi \Vdash \sigma \dashrightarrow \ell$ or $\pi \Vdash \ell \dashrightarrow \sigma$. Hence, $\pi \not\Vdash \sigma \diamond \ell$. 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. ◂

The proof of Thm. 24 will be conducted in the following Secs. D.1–D.4. It proceeds by induction on the structure of derivations in the SOS of $\mathsf{CCS}^{\mathsf{spt}}$, given in Fig. 2. We first show that the indirections of syntactic transformations via the static laws of Fig. 1, built into the SOS by the (*Struct*) rule, can in fact be eliminated in terms of a finite number of standard SOS rules that do not refer to structural laws.

▸ **Definition 57.** *A process P is called 0-free if either $P = 0$ or all occurrences of 0 in P are guarded by a prefix $\ell{:}H.R$.*

▸ **Proposition 58.** *Every process P is structurally congruent to a $\mathsf{0}$-free process Q.*

Proof. By application of the laws $P \star \mathsf{0} \equiv P$ and $\mathsf{0} \wr L \equiv \mathsf{0}$, where $\star \in \{\,|\,, +\}$ and $\wr \in \{\,\backslash\,, /\,\}$ we can eliminate every occurrence of $\mathsf{0}$ in an process P that is not of the form $\alpha{:}H.\mathsf{0}$. ◂

▸ **Proposition 59.** *If P is a $\mathsf{0}$-free process, and there is a derivation $\mathcal{D}$ for $P \xrightarrow[R]{\alpha}_H Q$, then all uses of the rule (Struct) in $\mathcal{D}$ can be eliminated in terms of the following symmetric versions of the rules (Par) and (Sum):*

$$\frac{P \xrightarrow[R]{\alpha}_H P'}{P + Q \xrightarrow[R]{\alpha}_H P'}\ (Sum_1) \qquad \frac{Q \xrightarrow[R]{\alpha}_H Q'}{P + Q \xrightarrow[R]{\alpha}_H Q'}\ (Sum_2)$$

$$\frac{P \xrightarrow[R]{\alpha}_H P' \quad \alpha \notin \mathcal{C}}{P \,|\, Q \xrightarrow[R \,|\, Q]{\alpha}_H P' \,|\, Q}\ (Par_1) \qquad \frac{Q \xrightarrow[R]{\alpha}_H Q' \quad \alpha \notin \mathcal{C}}{P \,|\, Q \xrightarrow[R \,|\, Q]{\alpha}_H P \,|\, Q'}\ (Par_2)$$

More specifically, there is a derivation $\mathcal{D}'$ for $P \xrightarrow[R']{\alpha}_H Q'$ with $R' \equiv R$ and $Q' \equiv Q$.

Proof Sketch. By induction on the structure of processes one shows that the transitions generated by the SOS with (*Struct*) can be simulated by the extended SOS without (*Struct*). The argument is based on the following case analysis:

- If $P = \mathsf{0}$ then no process structurally equivalent to P can generate any transition: If $P \equiv Q$ and $Q \xrightarrow[R]{\alpha}_H Q'$ then $P \neq \mathsf{0}$. Hence, the statement of the proposition holds trivially.
- For prefixes we show that if $P \equiv \ell{:}H.Q$ and $P \xrightarrow[E']{\ell'}_{H'} Q'$ then $\ell' = \ell$, $H' = H$, $E' \equiv \mathsf{0}$ and $Q' \equiv Q$.
- For sums we observe that if $P \equiv P_1 + P_2$ and $P \xrightarrow[E]{\ell}_H Q$ then there exists $i \in \{1, 2\}$ such that $P_i \xrightarrow[E']{\ell}_H Q'$ such that $E' \equiv E$ and $Q' \equiv Q$.
- For all other operators, e.g. $P \equiv P_1 \,|\, P_2$, $P \equiv Q \,/\, L$, $P \equiv Q \setminus L$, we can easily show that transitions $P \xrightarrow[R]{\alpha}_H P'$ can be obtained from transitions of P_i and Q.

◂

Note that the restriction in Prop. 59 is necessary. If P is not $\mathsf{0}$-free, say $P = \mathsf{0} \,|\, Q$, then we can have $\mathsf{0} \,|\, Q \xrightarrow{\sigma} Q'$ because of (*Struct*), because $\mathsf{0} \,|\, Q \equiv Q$ and $Q \xrightarrow{\sigma} Q'$, but without (*Struct*) we could not derive $\mathsf{0} \,|\, Q \xrightarrow{\sigma} Q'$.

Because of Prop. 58 we may assume our processes are $\mathsf{0}$-free. Because of Prop. 59 we may prove the preservation of coherence (for $\mathsf{0}$-free processes) by induction on the SOS rules without considering (*Struct*), but including the symmetric rules (Par_i) and (Sum_i) for $i \in \{1, 2\}$.

D.1 Stop and Prefixes

▸ **Proposition 60.** *The process $\mathsf{0}$ is coherent for any policy, i.e., $\mathsf{0} \Vdash \pi$ for all π.*

Proof. The inactive process $\mathsf{0}$ is locally coherent for trivial reasons, simply because it does not offer any transitions at all. There is no SOS rule applicable to it in the (*Struct*)-free setting. Note that in the system with (*Struct*) rule, we would have to argue that no process structurally equivalent to $\mathsf{0}$ has any transitions. ◂

▸ **Proposition 61.** *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 $\ell \in H \subseteq \{\ell' \mid \ell' \dashrightarrow \ell \in \pi\}$.*

Proof. A prefix expression $P = \ell{:}H.Q$ generates only a single transition by rule (*Act*), so that the assumption

$$P \xrightarrow[E_1]{\alpha_1}_{H_1} Q_1 \text{ and } P \xrightarrow[E_2]{\alpha_2}_{H_2} Q_2 \tag{1}$$

implies $\alpha_i = \ell$, $H_i = H$, $E_i \equiv 0$ and $Q_i \equiv Q$. It is easy to see that $\ell{:}H.Q$ conforms to π if $H \subseteq \{\beta \mid \beta \dashrightarrow \alpha \in \pi\}$. 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.

Again, it is important to notice that the SOS with (*Struct*) would not immediately warrant the conclusion that both derivations (1) are by (*Act*). This is because these transitions could be the transitions of two syntactically distinct (but structurally congruent) expressions $P_1 \equiv P \equiv P_2$. Instead, we would have to argue that if $P = \ell{:}H.Q \equiv P'$ and $P' \xrightarrow[E']{\alpha'}_{H'} Q'$ then $\alpha' = \ell$, $H' = H$, $E' \equiv 0$ and $Q' \equiv Q$. That this is the case is a property of our specific system of structural laws and handled by the above Prop. 59. This would obviously be violated, e.g., if we added a structural law like $\ell{:}H.Q \equiv \alpha'{:}H'.Q'$ where $\alpha' \neq \ell$, $H' \neq H$, $E' \not\equiv 0$ or $Q' \not\equiv Q$. ◂

As Examples 8 and 9 show, 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 62.** *If Q is coherent for π and $\sigma \in \mathcal{C}$ is a clock, then the prefix expression $\sigma{:}H.Q$ is coherent for π, too, if $H \subseteq \{\beta \mid \beta \dashrightarrow \sigma \in \pi\}$.*

Proof. The argument runs exactly as in case of Prop. 61. However, we do not need to require condition $\sigma \in H$ because if $\alpha_i = \sigma$ then $\alpha_1 = \alpha_2$, $Q_1 \equiv Q_2$ and $\{\alpha_1, \alpha_2\} \nsubseteq \mathcal{R}$. 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 63.** *Let $P_1 \Vdash \pi_1$ and $P_2 \Vdash \pi_2$ 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 rules (Sum_i). Conformance to π directly follows from the same property of Q or R, respectively. The proof of coherence Def. 12 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 $P_1 \not\equiv P_2$ 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 or 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. 12(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. 12(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 64** (Key lemma). *Let $Q \Vdash \pi$ and $R \Vdash \pi$ be such that $\pi \setminus \mathcal{R}$ is a pivot policy. Then $(Q \,|\, R) \Vdash \pi$.*

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 $Q \Vdash \pi$ or $R \Vdash \pi$ to conclude $\ell' \dashrightarrow \ell \in \pi$. If $\ell = \sigma$, then the transition in question is a synchronisation of Q and R via rule (Com)

$$Q \,|\, R \xrightarrow[0]{\sigma}_{H_2 \cup N_2 \cup B_2} Q_2 \,|\, R_2$$

with $E \equiv 0$, $H = H_2 \cup N_2 \cup B_2$ and $P' = Q_2 \,|\, R_2$ generated from transitions

$$Q \xrightarrow[0]{\sigma}_{H_2} Q_2 \text{ and } R \xrightarrow[0]{\sigma}_{N_2} R_2.$$

and $B_2 = \{\ell \,|\, \overline{\ell} \mid H_2 \cap \overline{\mathsf{iA}}(R) \not\subseteq \{\sigma\} \text{ or } N_2 \cap \overline{\mathsf{iA}}(Q) \not\subseteq \{\sigma\}\}$. By Lem. 51 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 $\ell' \dashrightarrow \ell \in \pi$ by conformance Def. 17 from $Q \Vdash \pi$ or $R \Vdash \pi$.

In the sequel, we tackle coherence Def. 12. As before, we work in the SOS without $(Struct)$ but with symmetrical rules (Par_i). 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 coherence of P and Q is preserved under transitions (i.e., by co-induction). Now suppose

$$P \xrightarrow[E_1]{\alpha_1}_{H_1} P_1 \text{ and } P \xrightarrow[E_2]{\alpha_2}_{H_2} P_2$$

such that $\alpha_1 \neq \alpha_2$ or $P_1 \not\equiv P_2$ 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 (Com). First note that by Lem. 51 we can drop the consideration of the synchronisation action $\ell \,|\, \overline{\ell}$ for the blocking set in (Com) altogether:

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

We will use the name (Com_a) to refer to the instance of (Com) with $\ell \in \mathcal{R}$ and the name (Com_σ) for the instance with $\ell \in \mathcal{C}$ and we proceeds by case analysis.

- $(Par_1, Par_2) \diamond (Com_a)$. 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 (Com_a). By symmetry, it suffices to consider the case that the non-synchronising transition is by (Par_1) from Q, i.e., $\alpha_1 \notin C$ and $\alpha_2 = \ell \,|\, \overline{\ell}$ and

$$Q \,|\, R \xrightarrow[F_1 \,|\, R]{\alpha_1} H_1 \; Q_1 \,|\, R \text{ and } Q \,|\, R \xrightarrow[F_2 \,|\, G_2]{\ell \,|\, \overline{\ell}} H_2' \cup N_2 \; Q_2 \,|\, R_2$$

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 rendezvous action such that

$$Q \xrightarrow[F_1]{\alpha_1} H_1 \; Q_1 \text{ and } Q \xrightarrow[F_2]{\ell} H_2' \; Q_2 \text{ and } R \xrightarrow[G_2]{\overline{\ell}} N_2 \; R_2.$$

We may safely assume $\alpha_1 \neq \ell \,|\, \overline{\ell} = \tau$ or $Q_1 \,|\, R \not\equiv Q_2 \,|\, R_2$. 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 $\{\alpha_1, \ell \,|\, \overline{\ell}\} \not\subseteq \mathcal{R}$ 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. 11(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. 44 and Lem. 45, this implies that $\ell \notin H_1$ up front. Next, if $\alpha_1 \neq \ell \,|\, \overline{\ell}$, non-interference Def. 11(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. 11(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. 11(1) for the diverging transitions out of Q.

For the second part of non-interference, suppose $\alpha_1 = \tau$. Then the non-interference assumption implies 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. 12 of Q and obtain processes F_1', F_2', Q' with reconverging transitions

$$Q_1 \xrightarrow[F_2']{\ell} H_2'' \; Q' \text{ and } Q_2 \xrightarrow[F_1']{\alpha_1} H_1' \; Q'$$

such that $H_2'' \subseteq H_2'$ and $H_1' \subseteq H_1$. We now invoke (Com_a) and (Par_1) to obtain reconverging transitions

$$Q_1 \,|\, R \xrightarrow[F_2' \,|\, G_2]{\ell \,|\, \overline{\ell}} H_2'' \cup N_2 \; Q' \,|\, R_2 \text{ and } Q_2 \,|\, R_2 \xrightarrow[F_1' \,|\, R_2]{\alpha_1} H_1' \; Q' \,|\, R_2$$

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 (Par_2) from R. Thus, we are looking at transitions

$$Q \xrightarrow[F_1]{\alpha_1} H_1 \; Q_1 \text{ and } R \xrightarrow[F_2]{\alpha_2} H_2 \; R_2$$

for $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R} \cup \{\tau\}$ combined via (Par) to generate

$$Q \,|\, R \xrightarrow[F_1 \,|\, R]{\alpha_1} H_1 \; Q_1 \,|\, R \text{ and } Q \,|\, R \xrightarrow[Q \,|\, F_2]{\alpha_2} H_2 \; Q \,|\, R_2$$

with $E_1 = F_1 \,|\, R$ and $E_2 = Q \,|\, F_2$. Then we can directly construct reconverging transitions, applying (Par) for

$$Q_1 \,|\, R \xrightarrow[Q_1 \,|\, F_2]{\alpha_2} H_2 \; Q_1 \,|\, R_2 \text{ and } Q \,|\, R_2 \xrightarrow[F_1 \,|\, R_2]{\alpha_1} H_1 \; Q_1 \,|\, R_2.$$

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 by (Par_2) from R). Then,

$$Q \,|\, R \xrightarrow[F_1 \,|\, R]{\alpha_1} H_1 \; Q_1 \,|\, R \text{ and } Q \,|\, R \xrightarrow[F_2 \,|\, R]{\alpha_2} H_2 \; Q_2 \,|\, R$$

with $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R} \cup \{\tau\}$ and $P_i = Q_i \,|\, R$, $E_i = F_i \,|\, R$, where

$$Q \xrightarrow[F_1]{\alpha_1} H_1 \; Q_1 \text{ and } Q \xrightarrow[F_2]{\alpha_2} H_2 \; Q_2$$

with $\alpha_1 \neq \alpha_2$ or $Q_1 \,|\, R \not\equiv Q_2 \,|\, R$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_i \notin H_{3-i}$. This, in particular, implies $Q_1 \not\equiv Q_2$. 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. 46. We can therefore apply coherence Def. 12 of Q and obtain processes F_1', F_2' and Q' with reconverging transitions

$$Q_1 \xrightarrow[F_2']{\alpha_2} H_2' \; Q' \text{ and } Q_2 \xrightarrow[F_1']{\alpha_1} H_1' \; Q'$$

with $H_i' \subseteq H_i$ and such that

$$F_1 \overset{\alpha_2}{\rightsquigarrow} F_1' \text{ and } F_2 \overset{\alpha_1}{\rightsquigarrow} F_2' \tag{2}$$

and if $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 \not\equiv Q_2$, more strongly

$$F_1 \xrightarrow{\alpha_2} F_1' \text{ and } F_2 \xrightarrow{\alpha_1} F_2'. \tag{3}$$

Reapplying (Par_1) we construct the transitions

$$Q_1 \,|\, R \xrightarrow[F_2' \,|\, R]{\alpha_2} H_2' \; Q' \,|\, R \text{ and } Q_2 \,|\, R \xrightarrow[F_1' \,|\, R]{\alpha_1} H_1' \; Q' \,|\, R.$$

Obviously, by (Par_1) and (Par_2) we also obtain transitions $F_1 \,|\, R \overset{\alpha_2}{\rightsquigarrow} F_1' \,|\, R$ and $F_2 \,|\, R \overset{\alpha_1}{\rightsquigarrow} F_2' \,|\, R$ from (2) or, more strongly, $F_1 \,|\, R \xrightarrow{\alpha_2} F_1' \,|\, R$ and $F_2 \,|\, R \xrightarrow{\alpha_1} F_2' \,|\, R$ from (3) if $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 \,|\, R \not\equiv Q_2 \,|\, R$.

- $(Com_a) \diamond (Com_a)$: 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 $P = Q \,|\, R$ and both the reductions

$$Q \,|\, R \xrightarrow[E_i]{\alpha_i} H_i \cup N_i \; Q_i \,|\, R_i$$

with $P_i = Q_i \,|\, R_i$ are τ-actions generated by the communication rule (Com), i.e. $\alpha_1 = \ell_1 \,|\, \overline{\ell_1} = \tau = \ell_2 \,|\, \overline{\ell_2} = \alpha_2$ 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 $P_1 \not\equiv P_2$, i.e., $Q_1 \not\equiv Q_2$ or $R_1 \not\equiv R_2$. Moreover, we only need a weak environment shift, because $\{\alpha_1, \alpha_2\} = \{\tau\} \not\subseteq \mathcal{R}$ and $\{\alpha_1, \alpha_2\} \cap \mathcal{C} = \{\,\}$. Also, non-interference Def. 11(2) means that $(H_i \cup N_i) \cap (\overline{\mathsf{iA}}^*(E_i) \cup \{\tau\}) = \{\,\}$, i.e.,

$$H_i \cap (\overline{\mathsf{iA}}^*(E_i) \cup \{\tau\}) = \{\,\} \text{ and } N_i \cap (\overline{\mathsf{iA}}^*(E_i) \cup \{\tau\}) = \{\,\}. \tag{4}$$

Thus, we are looking at transitions

$$Q \xrightarrow[F_i]{\ell_i}_{H_i} Q_i \text{ and } R \xrightarrow[G_i]{\overline{\ell}_i}_{N_i} R_i.$$

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. 11(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 (4). 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, $\ell_1 \diamond \ell_2 \in \pi$ or $\overline{\ell}_1 \diamond \overline{\ell}_2 \in \pi$, by Def. 48. Hence, $\ell_i \notin H_{3-i}$ or $\overline{\ell}_i \notin N_{3-i}$ by conformance Def. 17, 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. 12 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

$$Q_1 \xrightarrow[F_2']{\ell_2}_{H_2'} Q' \text{ and } Q_2 \xrightarrow[F_1']{\ell_1}_{H_1'} Q'.$$

such that $H_i' \subseteq H_i$. This means $\ell_i \in \mathsf{iA}(F_{3-i}) \subseteq \mathsf{iA}(F_{3-i} \,|\, G_{3-i}) = \mathsf{iA}(E_{3-i}) \subseteq \mathsf{iA}^*(E_{3-i})$ and therefore $\overline{\ell}_i \notin N_{3-i}$ because of (4). 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. 12 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

$$R_1 \xrightarrow[G_2']{\overline{\ell}_2}_{N_2'} R' \text{ and } R_2 \xrightarrow[G_1']{\overline{\ell}_1}_{N_1'} R'.$$

with $N_i' \subseteq N_i$. Finally, we invoke (*Com*) to obtain the re-converging reductions

$$Q_1 \,|\, R_1 \xrightarrow[F_2' \,|\, G_2']{\ell_2 \,|\, \overline{\ell_2}}_{H_2' \cup N_2'} Q' \,|\, R' \text{ and } Q_2 \,|\, R_2 \xrightarrow[F_1' \,|\, G_1']{\ell_1 \,|\, \overline{\ell_1}}_{H_1' \cup N_1'} Q' \,|\, R'$$

with $H_i' \cup N_i' \subseteq H_i \cup N_i$, and also

$$F_i \,|\, G_i \xrightarrow{\ell_{3-i} \,|\, \overline{\ell_{3-i}}} F_i' \,|\, G_i'$$

as required, since this implies the weaker form $F_i \,|\, G_i \overset{\tau}{\rightsquigarrow} F_i' \,|\, G_i'$ of environment shift.

It remains to consider the case that $\ell_1 = \ell = \ell_2$. By assumption $P_1 \not\equiv P_2$ and so we must have $Q_1 \not\equiv Q_2$ or $R_1 \not\equiv R_2$. Suppose $Q_1 \not\equiv Q_2$. If $R_1 \not\equiv R_2$ 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. 11, because $\ell_1 = \ell_2$ and $\ell_i \neq \tau$. Then, by coherence Def. 12 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 $Q_1 \not\equiv Q_2$)

$$Q_1 \xrightarrow[F_2']{\ell} {}_{H_2'} Q' \text{ and } Q_2 \xrightarrow[F_1']{\ell} {}_{H_1'} Q'.$$

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 (4). If $R_1 \equiv R_2$ then for each $i \in \{1,2\}$, by coherence Def. 12 (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 $G_i \equiv G_i'$ 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 $G_i \equiv G_i'$ 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 $R_1 \not\equiv R_2$ we can apply coherence Def 12 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 $R_1 \not\equiv R_2$) $G_1 \xrightarrow{\overline{\ell}} G_1'$ and $G_2 \xrightarrow{\overline{\ell}} G_2'$ as well as a process R' with

$$R_1 \xrightarrow[G_2']{\overline{\ell}} {}_{N_2'} R' \text{ and } R_2 \xrightarrow[G_1']{\overline{\ell}} {}_{N_1'} R'.$$

with $N_i' \subseteq N_i$. Finally, we invoke (Com) to obtain the re-converging reductions

$$Q_1 \,|\, R_1 \xrightarrow[F_2' \,|\, G_2']{\ell \,|\, \overline{\ell}} {}_{H_2' \cup N_2'} Q' \,|\, R' \text{ and } Q_2 \,|\, R_2 \xrightarrow[F_1' \,|\, G_1']{\ell \,|\, \overline{\ell}} {}_{H_1' \cup N_1'} Q' \,|\, R'$$

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.

- $(Com_a) \diamond (Com_\sigma)$ and, by symmetry, the induction case $(Com_\sigma) \diamond (Com_a)$: We assume $P = Q \,|\, R$ and the non-interfering, diverging reductions

$$Q \,|\, R \xrightarrow[E_1]{\ell \,|\, \overline{\ell}} {}_{H_1 \cup N_1 \cup B_1} Q_1 \,|\, R_1 \text{ and } Q \,|\, R \xrightarrow[0]{\sigma} {}_{H_2 \cup N_2 \cup B_2} Q_2 \,|\, R_2$$

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 (Com_a) and (Com_σ) of the communication rules, respectively. These transitions arise from

$$Q \xrightarrow[F_1]{\ell} {}_{H_1} Q_1 \text{ and } R \xrightarrow[G_1]{\overline{\ell}} {}_{N_1} R_1 \text{ and } Q \xrightarrow[0]{\sigma} {}_{H_2} Q_2 \text{ and } R \xrightarrow[0]{\sigma} {}_{N_2} R_2.$$

We assume that the c-actions $(\ell \,|\, \overline{\ell})$:$(H_1 \cup N_1 \cup B_1)[E_1]$ and σ:$(H_2 \cup N_2 \cup B_2)[0]$ are non-interfering. Note that $\ell \neq \sigma$ and $\overline{\ell} \neq \sigma$. First, observe that by Lem. 51, $B_1 = \{\} = B_2$. 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. 11(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. 11. But then, since $\{\ell, \sigma\} \cap \mathcal{C} \neq \{\,\}$, coherence Def. 12 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 (Com_a) and (Com_σ) are never interference-free.

- $\{(Par_1), (Par_2)\} \diamond (Com_\sigma)$: As the representative case, we assume $P = Q \,|\, R$ and the non-interfering, diverging reductions are

$$Q \,|\, R \xrightarrow[F_1 \,|\, R]{\alpha_1}_{H_1} Q_1 \,|\, R \text{ and } Q \,|\, R \xrightarrow[\mathsf{0}]{\sigma}_{H_2 \cup N_2 \cup B_2} Q_2 \,|\, R_2$$

with $\alpha_1 \in \mathcal{R} \cup \{\tau\}$ so that $P_1 \equiv Q_1 \,|\, R$ and $P_2 \equiv Q_2 \,|\, R_2$ are generated by the rule (Par_1) and (Com_σ), respectively, from transitions

$$Q \xrightarrow[F_1]{\alpha_1}_{H_1} Q_1 \text{ and } Q \xrightarrow[\mathsf{0}]{\sigma}_{H_2} Q_2 \text{ and } R \xrightarrow[\mathsf{0}]{\sigma}_{N_2} R_2.$$

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. 11(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. 12 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 (Com_σ) are never interference-free in the sense of coherence.

- $(Com_\sigma) \diamond (Com_\sigma)$: We assume $P = Q \,|\, R$ and the non-interfering, diverging reductions

$$Q \,|\, R \xrightarrow[\mathsf{0}]{\sigma_1}_{H_1 \cup N_1 \cup B_1} Q_1 \,|\, R_1 \text{ and } Q \,|\, R \xrightarrow[\mathsf{0}]{\sigma_2}_{H_2 \cup N_2 \cup B_2} Q_2 \,|\, R_2$$

with $P_i = Q_i \,|\, R_i$ and $E_i = F_i \,|\, G_i$ are generated by the communication rules (Com_σ). Observing that $\{\sigma_1, \sigma_2\} \subseteq \mathcal{R}$ is impossible, here, we may assume $\sigma_1 \neq \sigma_2$ or $Q_1 \,|\, R_1 \not\equiv Q_2 \,|\, R_2$. This can only hold true if $Q_1 \not\equiv Q_2$ or $R_1 \not\equiv R_2$. 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 $Q \xrightarrow[\mathsf{0}]{\sigma}_{H_1} Q_1$ and $Q \xrightarrow[\mathsf{0}]{\sigma}_{H_2} Q_2$ and the likewise the transitions $R \xrightarrow[\mathsf{0}]{\sigma}_{N_1} R_1$ and $R \xrightarrow[\mathsf{0}]{\sigma}_{N_2} R_2$ are interference-free. If $Q_1 \not\equiv Q_2$ then we could apply coherence of Q and obtain $\sigma \in \mathsf{iA}(\mathsf{0})$, if $R_1 \not\equiv R_2$ the same follows from coherence of R. This is impossible, whence the case can be excluded. ◂

D.4 Hiding

▸ **Proposition 65.** *If Q is coherent for π and $\sigma \dashrightarrow \ell \notin \pi$ for all $\sigma \in L$ and $\ell \in \pi$, 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:

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

where $\sigma\,/\,\sigma = \tau$ and $\alpha\,/\,\sigma = \alpha$ if $\alpha \neq \sigma$. Let $P = Q\,/\,\sigma$ and the two non-interfering and diverging transitions

$$P \xrightarrow[E_1]{\alpha_1\,/\,\sigma}_{H_1} P_1 \text{ and } P \xrightarrow[E_2]{\alpha_2\,/\,\sigma}_{H_2} P_2$$

arise by (*Hide*) from transitions

$$Q \xrightarrow[F_1]{\alpha_1}_{N_1} Q_1 \text{ and } Q \xrightarrow[F_2]{\alpha_2}_{N_2} Q_2$$

with $E_i = F_i\,/\,\sigma$, $P_i = Q_i\,/\,\sigma$ and $H_i = N_i - \{\sigma\}$. For coherence Def. 12 we assume that $\alpha_1\,/\,\sigma \neq \alpha_2\,/\,\sigma$ or $P_1 \not\equiv P_2$, 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 $Q_1 \not\equiv Q_2$. Moreover, because of the assumption that $\sigma \dashrightarrow \alpha_i \notin \pi$ 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 $Q_1 \not\equiv Q_2$ contradicts strong determinacy of clocks (Proposition 15). 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. 54 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. 12 for Q by induction hypothesis and get F_1', F_2', Q' such that

$$Q_1 \xrightarrow[F_2']{\alpha_2}_{N_2'} Q' \text{ and } Q_2 \xrightarrow[F_1']{\alpha_1}_{N_1'} Q' \text{ with } F_1 \overset{\alpha_2}{\rightsquigarrow} F_1' \text{ and } F_2 \overset{\alpha_1}{\rightsquigarrow} F_2'$$

such that $N_i' \subseteq N_i$. By applying (*Hide*) to these transitions, we obtain

$$P_1 \xrightarrow[F_2'\,/\,\sigma]{\alpha_2\,/\,\sigma}_{H_2''} Q'\,/\,\sigma \text{ and } P_2 \xrightarrow[F_1'\,/\,\sigma]{\alpha_1\,/\,\sigma}_{H_1''} Q'\,/\,\sigma$$

where $H_i'' = N_i' - \{\sigma\} \subseteq N_i - \{\sigma\} = H_i$. Note, by rule (*Hide*) we also have

$$F_1\,/\,\sigma \overset{\alpha_2\,/\,\sigma}{\rightsquigarrow} F_1'\,/\,\sigma \text{ and } F_2\,/\,\sigma \overset{\alpha_1\,/\,\sigma}{\rightsquigarrow} F_2'\,/\,\sigma.$$

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 $Q_1\,/\,\sigma \not\equiv Q_2\,/\,\sigma$, 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

$$F_1\,/\,\sigma \xrightarrow{\alpha_2\,/\,\sigma} F_1'\,/\,\sigma \text{ and } F_2\,/\,\sigma \xrightarrow{\alpha_1\,/\,\sigma} F_2'\,/\,\sigma.$$

This completes the proof. ◂

D.5 Restriction

Let us next look at the restriction operator. The following definition is relevant for creating coherent processes under action restriction (App. D.5).

The fact that P is locally coherent does not imply that $P \setminus L$ is locally coherent. Consider the process $Q = (s + a{:}s \mid b.\overline{s}) \setminus s$ 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

$$Q \xrightarrow[(1 \mid b.\overline{s}) \setminus s]{a}{}_{\{\,\}} (b.\overline{s}) \setminus s$$

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 $Q \mid \overline{b}$ will internally set the sender $\overline{s}$ free and thus block the a-transition. From the above transition the (*Com*) rule permits $Q \mid \overline{b}$ to offer the a-step. This is not right as it is internally blocked and does not commute with the $b \mid \overline{b}$ 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. Another, more general solution is to use the extended rule *Restr** discussed above after Example 34.

▸ **Proposition 66.** *If $Q \Vdash \pi$ and π is precedence-closed for $L \cup \overline{L}$, then $(Q \setminus L) \Vdash \pi \setminus L$.*

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

$$Q \setminus a \xrightarrow[E \setminus a]{\ell}_{H'} Q' \setminus a \text{ derived from } Q \xrightarrow[E]{\ell}_H Q'$$

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. 17 applied to Q implies $\ell' \dashrightarrow \ell \in \pi$. But then $\ell' \dashrightarrow \ell \in \pi - a$, 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 $E_i \equiv F_i \setminus a$ and $P_i \equiv Q_i \setminus a$. We assume $\alpha_1 \neq \alpha_2$ or $P_1 \not\equiv P_2$ 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 $P_1 \not\equiv P_2$ we immediately have $Q_1 \not\equiv Q_2$. 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 $\ell \dashrightarrow \alpha_i \in \pi$. 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. 12 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. 11(1) to show that the c-actions $\alpha_i{:}N_i[F_i]$ are interference-free. For the second part Def. 11(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' such that

$$Q_1 \xrightarrow[F_2']{\alpha_2}_{N_2'} Q' \text{ and } Q_2 \xrightarrow[F_1']{\alpha_1}_{N_1'} Q' \text{ with } F_1 \overset{\alpha_2}{\rightsquigarrow} F_1' \text{ and } F_2 \overset{\alpha_1}{\rightsquigarrow} F_2'$$

such that $N_i' \subseteq N_i$. We construct the required reconvergence by application of rule (*Restr*):

$$P_1 \xrightarrow[E_2']{\alpha_2}_{H_2''} Q' \setminus a \text{ and } P_2 \xrightarrow[E_1']{\alpha_1}_{H_1''} Q' \setminus a \text{ with } F_1 \setminus a \overset{\alpha_2}{\rightsquigarrow} F_1' \setminus a \text{ and } F_2 \setminus a \overset{\alpha_1}{\rightsquigarrow} F_2' \setminus a$$

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 $P_1 \not\equiv P_2$ 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. ◂