



HAL
open science

Strong Priority and Determinacy in Timed CCS

Luigi Liquori, Michael Mendler

► **To cite this version:**

Luigi Liquori, Michael Mendler. Strong Priority and Determinacy in Timed CCS. Inria; University of Bamberg. 2023. hal-04367635v2

HAL Id: hal-04367635

<https://inria.hal.science/hal-04367635v2>

Submitted on 29 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

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* $\bar{\alpha}$ 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 $S \stackrel{\text{def}}{=} r.S + w.\bar{r}.0$ acts as a shared resource, such as a (write-once) store, while $R \stackrel{\text{def}}{=} \bar{r}.0$, $W \stackrel{\text{def}}{=} \bar{w}.0$ 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 $\bar{r}.0$. 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 \bar{r} , \bar{w} generates two sequences of reductions from $R | S | W$:

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



© Author: Please provide a copyright holder;

licensed under Creative Commons License CC-BY 4.0

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

XX:2 Strong Priority and Determinacy in Timed CCS

43 Obviously, $R | S | W$ is not determinate. It reduces to two distinct normal forms $0 | \bar{r}.0 | 0$ and
44 $0 | 0 | 0$. The final store $\bar{r}.0$ on the left permits one more read, while 0 on the right does not.

45 A sufficient condition for determinacy is determinism. A process P is (*structurally*)
46 *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
47 $Q_1 \equiv Q_2$, where \equiv is a suitable structural congruence. However, determinism is too strong.
48 For instance, the store $P \stackrel{def}{=} R_1 | S | R_2$ with two readers $R_i \stackrel{def}{=} \bar{r}.R'_i$ ($i=1, 2$) is determinate
49 but not deterministic, when the R_i may reach structurally distinct states after reading, say,
50 $P \xrightarrow{\tau} R'_1 | S | R_2$ and $P \xrightarrow{\tau} R_1 | S | R'_2$ where $R'_1 | S | R_2 \not\equiv R_1 | S | R'_2$. Moreover, determinism
51 is not closed under parallel composition. In the above example, all process R_1, R_2, S are
52 deterministic, but their composition P is not. An insightful solution, proposed by Milner
53 (Chap. 11 of [27]), is to replace determinism by the notion of *confluence* which still implies
54 determinacy but turns out to be closed under parallel composition, under natural restrictions.

55 **► Definition 1.** P is (*structurally*) *confluent*¹ if for every derivative Q of P and reductions
56 $Q \xrightarrow{\alpha_1} Q_1$ 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 $Q_1 \xrightarrow{\alpha_2} Q'_1$
57 and $Q_2 \xrightarrow{\alpha_1} Q'_2$.

58 Milner shows that confluence is preserved by “*confluent composition*” $P |_L Q \stackrel{def}{=} (P | Q) \setminus L$,
59 combining the parallel and restriction operators, subject to the condition that $\mathcal{L}(P) \cap \mathcal{L}(Q) =$
60 $\{\}$ and $\overline{\mathcal{L}(P)} \cap \mathcal{L}(Q) = L \cup \bar{L}$, where $\mathcal{L}(R)$ is the *sort* of a process R . This is a form of *sort*
61 *separation*, ensuring that every action α has at most one synchronisation partner $\bar{\alpha}$ either
62 inside or outside of $(P | Q) \setminus L$. Many practical examples of determinate systems can be
63 understood as sort-separated compositions of confluent processes. However, this still covers
64 only a rather limited class of applications. Most seriously, memory processes such as S above
65 are intrinsically not confluent, and even if they were, confluent composition forbids direct
66 multi-cast communication, because labels such as r in S could not be shared by two readers
67 R_1, R_2 due to sort-separation. This means that concurrent programming languages that
68 support shared memory and yet have determinate reduction semantics, cannot be handled.

69 Determinacy in Synchronous Programming

70 The most prominent class of shared-memory programming languages that manage to reconcile
71 concurrency and determinacy, is known as *Synchronous Programming* (SP). In SP, processes
72 interact with each other asynchronously at a *micro-step* level, yet synchronise in lock-step to
73 advance jointly in iterative cycles of synchronous *macro-steps*. The micro-step interactions
74 taking place during a macro-step determine the final outcome of the macro-step. This
75 outcome is defined at the point where all subsystem pause to wait for a global logical *clock*.
76 When this clock arrives, all subsystem proceed into the next computation cycle. Under the
77 so-called *synchrony hypothesis*, the final outcome of a macro-step for each subsystem is fully
78 determined from the stimulus provided by the environment of that component during the
79 micro-step interactions. As such, the interactions inside each subsystem at macro-step level
80 can be abstracted into a global Mealy automaton with deterministic I/O. SP started with
81 Statecharts [17] and has generated languages such as Signal [14], Lustre [16], Esterel [5],
82 Quartz [37], SCCharts [41], just to mention a few.

83 When it comes to mathematical modelling it is natural to think of a clock as a broadcast
84 action in the spirit of Hoare’s CSP [19] that acts as a synchronisation barrier for a set of
85 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 .

86 into a sequence of macro-steps that are aligned with each other in a lock-step fashion. During
 87 each macro-step, a process executes only a temporal slice of its total behaviour, at the end of
 88 which it waits for all other processes in the same *clock scope* to reach the end of the current
 89 phase. When all processes have reached the barrier, they are released into the next round
 90 of computation. This suggests a priority-based scheduling mechanism, i.e., that the clock
 91 should fire only if the system has stabilised and no other admissible data actions are possible.
 92 This principle, called *maximal progress*, is built into timed process algebras, see e.g. [18, 11].

93 Contribution

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

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

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

XX:4 Strong Priority and Determinacy in Timed CCS

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

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

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

2 A Process Algebra with Clocks and Priorities

148 The syntax and operational semantics of our process algebra ‘SynPaTick’, denoted CCS^{spt} , is
 149 defined in this section.

2.1 CCS^{spt} Syntax

151 The terms in CCS^{spt} form a set \mathcal{P} of *processes*. We use $P, Q, R, S \dots$ to range over \mathcal{P} . Let
 152 \mathcal{I} be a set of *process names* and let $A, B \dots$ range over \mathcal{I} . Let \mathcal{A} be a countably infinite
 153 set of *channel names* with $a, b \dots$ ranging over \mathcal{A} . As in CCS, the set $\bar{\mathcal{A}}$ is a disjoint set
 154 of *co-names* with elements denoted by $\bar{a}, \bar{b}, \bar{c} \dots$ in bijection with \mathcal{A} ; The overbar operator
 155 switches between \mathcal{A} and $\bar{\mathcal{A}}$, i.e., $\bar{\bar{a}} = a$ for all $\bar{a} \in \bar{\mathcal{A}}$. We will refer to the names $a \in \mathcal{A}$ as
 156 *input* (or *receiver*) labels while the co-names $\bar{a} \in \bar{\mathcal{A}}$ denote *output* (or *sender*) labels. Let
 157 \mathcal{C} be a countably infinite set of broadcasted *clock names*, disjoint from both \mathcal{A} and $\bar{\mathcal{A}}$ and
 158 let σ range over \mathcal{C} . Every clock acts as its own co-name, $\bar{\sigma} = \sigma$ and so each $\sigma \in \mathcal{C}$ is also
 159 considered as a *clock label*. Let $\mathcal{L} = \mathcal{A} \cup \bar{\mathcal{A}} \cup \mathcal{C}$ be the set of input, output and clock labels
 160 and let ℓ range over \mathcal{L} , while L, H range over subsets of \mathcal{L} . We write \bar{L} for the set $\{\bar{\ell} \mid \ell \in L\}$.
 161 Let $\text{Act} = \mathcal{L} \cup \{\tau\}$ be the set of all *actions* obtained by adjoining the *silent action* $\tau \notin \mathcal{L}$ and
 162 let α range over Act . Viewed as actions, the input and output labels $\ell \in \mathcal{R} \stackrel{\text{def}}{=} \mathcal{A} \cup \bar{\mathcal{A}} \subseteq \text{Act}$
 163 are referred to as *rendezvous actions* to distinguish them from the *clock actions* $\sigma \in \mathcal{C} \subseteq \text{Act}$.
 164 All symbols can appear indexed. The process terms \mathcal{P} are defined thus:

165	$P, Q, R, S ::=$	0	stop (inaction)		$P \mid Q$	parallel	
			A	name, $A \in \mathcal{I}$		$P \setminus L$	restriction
			$\alpha:H.P$	action, $\alpha \in \mathcal{L}, H \subseteq \mathcal{L}$		P/L	hiding
166			$P + Q$	choice		P/L	hiding

167 Intuitively, 0 denotes the *inactive* process which stops all computations, including clock
 168 actions. The *action prefix* $\alpha:H.P$ denotes a process that offers to engage in the action α
 169 and then behaves as P while the *blocking set* $H \subseteq \mathcal{L}$ lists all actions taking precedence over
 170 α . A prefix with $\alpha \in \mathcal{R}$ denotes a CCS-style rendezvous action. In case $\alpha \in \mathcal{C}$, it denotes a
 171 CSP-style broadcast action that can communicate with all the surrounding processes sharing
 172 on the same clock only when the clock is universally *consumed*. We assume that in every
 173 restriction $P \setminus L$ the set $L \subseteq \mathcal{R}$ consists of rendezvous actions, and in every hiding P/L
 174 we have $L \subseteq \mathcal{C}$. These *scoping* operators act as name binders that introduce local scopes.

$$\begin{array}{ll}
\text{(Comm)} & P \star Q \equiv Q \star P \\
\text{(Assoc)} & (P \star Q) \star R \equiv P \star (Q \star R) \\
\text{(Zero)} & P \star 0 \equiv P \\
\text{(Scope}_\alpha\text{)} & (\alpha:H.P) \setminus L \equiv \begin{cases} 0 & \text{if } \alpha \in L \cup \bar{L} \\ \alpha:H.(P \setminus L) & \text{otherwise} \end{cases} \\
\text{(Scope}_\setminus\text{)} & (P \star Q) \setminus L \equiv \begin{cases} P \star Q \setminus L & \text{if } \mathcal{L}(P) \cap (L \cup \bar{L}) = \{\} \\ P \setminus L + Q \setminus L & \text{if } \star = + \\ P \setminus L | Q \setminus L & \text{if } \star = | \text{ and } \mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} \cap (L \cup \bar{L}) = \{\} \end{cases} \\
\text{(Idem}_+\text{)} & P + P \equiv P \\
\text{(Scope}_\setminus\text{)} & P \setminus L \setminus L' \equiv P \setminus (L \cup L') \\
\text{(Scope}_0\text{)} & 0 \setminus L \equiv 0
\end{array}$$

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

175 The acute reader will notice the absence of the relabelling operator $P[f]$, useful in CCS
176 to define the well-known “standard concurrent form” $(P_1[f_1] | \dots | P_2[f_n]) \setminus L$. Because all
177 of our examples are captured without need of relabelling, and for the sake of simplicity,
178 relabelling will be treated in the full version of this work. A useful abbreviation is to drop
179 the blocking sets if they are empty or drop the continuation process if it is inactive. For
180 instance, we will typically write $a.P$ instead of $a:\{\}.P$ and $a:H$ rather than $a:H.0$. Also, it
181 is useful to drop the braces for the blocking set if it is a singleton, writing $a:b.P$ instead of
182 $a:\{b\}.P$. In our concrete notation using the abstract syntax of processes, we assume that
183 the binary operator $+$ is right associative and taking higher binding strength than the binary
184 operator $|$, i.e., writing $P + Q + R | S$ instead of $(P + (Q + R)) | S$. Also, we assume that the
185 unary operators of prefix, restriction and hiding have higher binding strength than the binary
186 operators. Hence, we write $P \setminus c + a.b.c.Q | R / \sigma$ instead of $((P \setminus c) + (a.(b.(c.Q)))) | (R / \sigma)$.
187 The *sub-processes* of a P are all sub-expressions of P and (recursively) all sub-processes of Q
188 for *definitions* $A \stackrel{\text{def}}{=} Q$ where A is a sub-expression of P . Sometimes it is useful to consider
189 fragments of \mathcal{P} . Specifically, a process is called *unclocked* if P does not contain any clock
190 prefix $\sigma:H.Q$ as a sub-process. A process is called *sequential* or *single-threaded* if it does not
191 contain any parallel sub-process $Q | R$. P is *closed* if all sub-processes $\ell:H.Q$ in P occur in
192 the scope of a restriction or hiding operator that binds ℓ . As usual, we denote by $\mathcal{L}(P) \subseteq \mathcal{L}$
193 the free labels, called (*syntactic*) *sort* of P , identify processes that differ only in the choice of
194 bound labels.

195 Following Milner’s presentation of the π -calculus in [28], we define a structural congruence
196 over processes. Specifically, *structural congruence* \equiv is the smallest congruence over \mathcal{P} that
197 satisfies the equations in Fig. 1. The idea of the structural congruence is to split interaction
198 rules between agents from rules governing the “left-to-right” representation of expressions. It
199 allows to use those equations in the derivation rules of the SOS making the latter simpler and
200 easy to use, and it allows to set up formal equivalence of processes increasing for each program
201 the number of legal SOS transitions that, otherwise, would be blocked. Milner divides CCS
202 equations in static, dynamic and expressions laws. We choose an approach inspired by
203 the work of Berry and Boudol [7] by adding an explicit structural rule in the operational
204 semantics. Note that we do not include in our structural rules all of Milner’s τ -laws, i.e. the
205 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
206 have also dropped the congruence equations related to the uniqueness of solution of recursive
207 equations and the *expansion laws* related to the *standard concurrent form* (cf. Chap. 3,
208 Props. 2, 4(2) and 5 of [27]). Since we do not propose notions of behavioural equivalence in
209 this paper, we prefer to take a conservative approach and only consider a minimum set of
210 purely structural rules that do not impinge on the dynamics of expression behaviour.

2.2 Operational Semantics

The operational semantics for CCS^{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 $\alpha \in \text{Act}$ 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, $\alpha \in \text{Act}$ is an action, and $H \subseteq \text{Act}$ 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 CCS^{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 $A \stackrel{\text{def}}{=} P$.

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

$$\begin{array}{c}
\frac{}{\alpha:H.P \xrightarrow[0]{\alpha}_H P} \text{ (Act)} \qquad \frac{P \equiv P' \quad P' \xrightarrow[R']{\alpha}_H Q' \quad Q' \equiv Q \quad R' \equiv R}{P \xrightarrow[R]{\alpha}_H Q} \text{ (Struct)} \\
\frac{P \xrightarrow[R]{\alpha}_H Q \quad L' = L \cup \bar{L} \quad \alpha \notin L' \quad H' = H - L'}{P \setminus L \xrightarrow[R \setminus L]{\alpha}_H Q \setminus L} \text{ (Restr)} \qquad \frac{P \xrightarrow[R]{\alpha}_H Q \quad H' = H - L}{P / L \xrightarrow[R/L]{\alpha/L}_H Q / L} \text{ (Hide)} \\
\frac{P \xrightarrow[R]{\alpha}_H P' \quad \alpha \notin C}{P | Q \xrightarrow[R|Q]{\alpha}_H P' | Q} \text{ (Par)} \qquad \frac{P \xrightarrow[R]{\alpha}_H P'}{P + Q \xrightarrow[R]{\alpha}_H P'} \text{ (Sum)} \\
\frac{P \xrightarrow[R_1]{\ell}_H P' \quad Q \xrightarrow[R_2]{\bar{\ell}}_H Q' \quad H = \text{race}(P, Q)}{P | Q \xrightarrow[R_1 | R_2]{\ell | \bar{\ell}}_H P' | Q'} \text{ (Com)} \qquad \frac{A \stackrel{\text{def}}{=} P \quad P \xrightarrow[R]{\alpha}_H P'}{A \xrightarrow[R]{\alpha}_H P'} \text{ (Con)}
\end{array}$$

where

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

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

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

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

258 Let us call a process P *free* if all its c-actions $\alpha:H[R]$ have an empty blocking set $H = \{\}$.
259 The weakly enabled transitions of (unlocked) free processes coincide with the semantics of
260 CCS [27]. We call P *discrete* if all its c-actions $\alpha:H[R]$ are self-blocking, i.e. $H = \{\alpha\}$. A
261 process P is *irreflexive* if no action prefix $\alpha:H.Q$ occurring in P blocks itself, i.e., $\alpha \notin H$.
262 One can show that for (unlocked) irreflexive processes, a transition is weakly enabled iff
263 is derivable in the operational semantics of CCS^{Ph} [33]². Thus, CCS and CCS^{Ph} correspond
264 to theories of free and irreflexive processes of CCS^{spt} , respectively, under weak enabling.
265 Likewise the theories [10, 36] can be subsumed; specifically, the *prioritised sum* operator
266 $P \dot{\rightarrow} Q$ of [10] can be coded as $P + Q:iA(P)$ in CCS^{spt} . The theory [36] proposes two *levels* of
267 labels, $a:0$ and $a:1$, where 0 denotes higher priority and 1 ordinary priority. Translated into
268 CCS^{spt} , the prefix $a:0.P$ corresponds to $a:\{\}$. P while an action prefix $a:1.P$ maps in CCS^{spt}
269 to $a:\text{Prio}.P$ if we define Prio as the set of all high-prioritised labels.

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

277 ▶ **Definition 3** (Potential actions and constructive enabling).

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

280 ■ A transition with c-action $\alpha:H[R]$ is called *constructively enabled*, or *c-enabled for short*,
281 if $H \cap (\bar{iA}^*(R) \cup \{\tau\}) = \{\}$.

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

XX:8 Strong Priority and Determinacy in Timed CCS

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

3 Examples

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

287 ▶ **Example 4** (Read Before Write). Consider a “store” $S \stackrel{\text{def}}{=} w.r + r:w$ that offers a write action
288 w followed by a read r , or a read action r but prefers the write over the read. The fact that
289 w takes priority over r is recorded in the blocking set $\{w\}$ which contains the action w . If we
290 put S in parallel with a reader $R \stackrel{\text{def}}{=} \bar{r}$, the read actions r and \bar{r} synchronise in a reduction
291 $S | R \xrightarrow{\tau}_{\{w\}} 0$. The blocking set $\{w\}$ will block R ’s transition when a parallel writer $W \stackrel{\text{def}}{=} \bar{w}$ is
292 added, as well. Specifically, the reduction $S | R | W \xrightarrow{\tau}_{\{w\}} W$ is not weakly enabled, because
293 $\{w\} \cap \bar{iA}(\bar{w}) = \{w\} \cap \{w\} \neq \{\}$. The only enabled reduction is that between S and W giving
294 $S | R | W \xrightarrow{\tau}_{\{w\}} r | R$ and only then continue with the read $r | R \xrightarrow{\tau}_{\{w\}} 0$. Thus, weak enabling
295 makes the reduction determinate. Observe that S is not confluent (Def. 1): We have $S \xrightarrow{w} r$
296 and $S \xrightarrow{\tau} 0$ but there is no $P_1 \equiv P_2$ such that $r \xrightarrow{\tau} P_1$ and $0 \xrightarrow{w} P_2$. However, S is coherent
297 (Def. 12), because the transition $S \xrightarrow{\tau}_{\{w\}} 0$ interferes (Def. 11) with the former $S \xrightarrow{w}_{\{w\}} r$ as
298 $w \in \{w\}$, whence coherence does not require confluence.

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

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

310 ▶ **Example 6** (Reflexive Blocking). Consider the free process $s | \bar{s}.A | \bar{s}.B$ where action s can
311 be consumed by $\bar{s}.A$ or by $\bar{s}.B$, generating non-deterministic τ -transitions to $A | \bar{s}.B$ or to
312 $\bar{s}.A | B$. However, by adding self-blocking, as in $s:s | \bar{s}.A | \bar{s}.B$, weak enabling protects the
313 prefix s from being consumed when both $\bar{s}.A$ and by $\bar{s}.B$ compete for it. If only one is
314 present, as in $s:s | \bar{s}.A$ and $s:s | \bar{s}.B$, no blocking occurs.

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

319 ▶ **Example 7** (Transitive Blocking). Consider the process $S \stackrel{\text{def}}{=} w_0 + r_0:w_0 | w_1 + r_1:w_1$ which
320 offers receiver actions w_i and r_i for $i \in \{0, 1\}$ such that w_i has precedence over r_i . Now
321 take processes $P_0 \stackrel{\text{def}}{=} \bar{r}_0.\bar{w}_1$ and $P_1 \stackrel{\text{def}}{=} \bar{r}_1.\bar{w}_0$ which first aim to synchronise with r_i and then
322 sequentially afterwards with w_{1-i} . The parallel composition $P_0 | S | P_1$ has two initial τ -re-
323 ductions $P_0 | w_0 + r_0:w_0 | \bar{w}_0 \xrightarrow{\tau}_{P_0 | (w_0 + r_0:w_0) \{w_1\}} P_0 | S | P_1 \xrightarrow{\tau}_{w_1 + r_1:w_1 | P_1 \{w_0\}} \bar{w}_1 | w_1 + r_1:w_1 | P_1$.

324 These reductions are both weakly enabled. If we continue under weak enabling then the
 325 residual process $\bar{w}_1 | w_1 + r_1 : w_1 | P_1$ for the first reduction must first execute the synchronisation
 326 on w_1 and thus proceed as $\bar{w}_1 | w_1 + r_1 : w_1 | P_1 \xrightarrow{\tau} P_1$ while the residual process
 327 $P_0 | w_0 + r_0 : w_0 | \bar{w}_0$ can only reduce to P_0 . Hence, the resulting final outcome of executing
 328 $P_0 | S | P_1$ is non-deterministic. Observe that none of the two reductions displayed above
 329 is actually c-enabled. For the right reduction we find that its concurrent environment
 330 $R_0 \stackrel{def}{=} w_1 + r_1 : w_1 | P_1$ has a reduction sequence $R_0 \xrightarrow{\tau} \bar{w}_0$ and thus $\{w_0\} \cap \bar{iA}^*(R_0) \subseteq$
 331 $\{w_0\} \cap \bar{iA}^*(\bar{w}_0) = \{w_0\} \cap \{w_0\} \neq \{\}$. Symmetrically, for the concurrent environment $R_1 \stackrel{def}{=} P_0 | w_0 + r_0 : w_0$
 332 of the left reduction we have $R_1 \xrightarrow{\tau} \bar{w}_1$ and hence $\{w_1\} \cap \bar{iA}^*(R_1) \neq \{\}$. The
 333 parallel composition $P_0 | S | P_1$ is deterministic under c-enabling. We note again, S is not
 334 confluent but coherent (Def. 12).

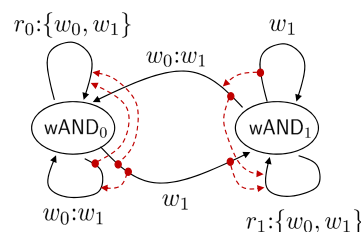
335 The next example generalises Ex. 4 to a full memory cell.

336 ▶ **Example 8 (Read/Write Memory)**. Let w_0, w_1, r_0, r_1 the labels of writing/reading a boolean
 337 value 0 or 1, respectively. A memory would be modelled by two mutually recursive processes

$$338 \quad \text{wAND}_0 \stackrel{def}{=} w_1.\text{wAND}_1 + w_0:w_1.\text{wAND}_0 + r_0:\{w_0, w_1\}.\text{wAND}_0$$

$$339 \quad \text{wAND}_1 \stackrel{def}{=} w_1.\text{wAND}_1 + w_0:w_1.\text{wAND}_0 + r_1:\{w_0, w_1\}.\text{wAND}_1$$

340 The process wAND_v represents the memory cell in a state where it has stored the value $v \in \{0, 1\}$. In either
 341 state, it offers to synchronise with a write action w_u to change its state to wAND_u , and also with a read action
 342 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,
 343 they also block the read actions since they appear in their blocking sets $r_v:\{w_0, w_1\}$.



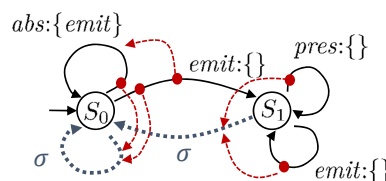
341 Between the write actions, the blocking sets implement a precedence of w_1 over w_0 . In the
 342 picture on the right the process state transitions are visualised with the blocking sets as red
 343 dashed arrows.

344 ▶ **Example 9 (Esterel Signals)**. Concurrent Esterel threads communicate via *signals* [6, 38].
 345 A (pure, temporary) signal can have two statuses, *present* and *absent*. At the beginning
 346 of a synchronous macro-step, each signal is initialised to be absent, by default. The signal
 347 becomes present as soon as some thread emits it. It remains present thereafter for throughout
 348 the current macro-step, i.e., until the next clock cycle is started. The clock is a global
 349 synchronisation, taken by all threads simultaneously. The value of the signal is broadcast to
 350 all concurrent threads which can test it and branch their control-flow according to the signal's
 351 status. To maintain coherency of data-flow and deterministic behaviour, the signal status
 352 can only be read when no writing is possible any more. Esterel compilers conduct a causality
 353 analysis on the program and obtain a suitable static schedule when they generate imperative
 354 code. In Esterel hardware compilers, the scheduling is achieved dynamically by propagating
 355 signal statuses. Programs that cannot be scheduled to satisfy the emit-before-test protocol,
 356 or hardware that that does not stabilise are called *non-constructive* and rejected by the
 357 compiler. A pure, temporary signal is modelled by the following recursive behaviour:

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

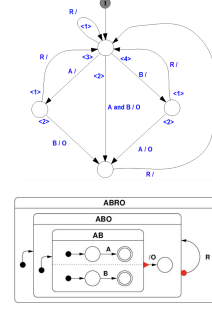
$$S_1 \stackrel{def}{=} \text{pres}.S_1 + \text{emit}.S_1 + \sigma:\{\text{pres}, \text{emit}\}.S_0$$

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



XX:10 Strong Priority and Determinacy in Timed CCS

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



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

359 ▶ **Example 10 (ABRO).** The *hello word* in synchronous languages, see [6], ABRO has the
 360 following behaviour: *Emit the output o as soon as both inputs a and b have been received.*
 361 *Reset this behaviour each time the input r occurs, without emitting o in the reset cycle.* [38]
 362 The behaviour of ABRO is expressed as a Mealy machine and a SyncChart, displayed in Fig. 3
 363 on the right. The synchronous behaviour of Esterel lies in the assumption that each transition
 364 of the Mealy machine summarises a single *macro-step* interaction of the machine with its
 365 environment. The transitions of parallel machines synchronise so that state changes proceed
 366 in lock-step. The simultaneous execution of these transitions that make up a macro-step
 367 consists of the sending and receiving of signals. The propagation of individual signals between
 368 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.

369 The loop `P each R` construct of Esterel executes the behaviour of program `P` for an
 370 unbounded number of clock cycles, but restarts its body `P` when the signal `R` becomes
 371 present. The reset `R` takes priority over the behaviour of `P` which is preempted at the moment
 372 that the reset occurs. Note that in Esterel each signal appears exactly once, as in the
 373 specification and unlike in the Mealy machine. Other parts of Esterel also naturally arise
 374 from precedence-based scheduling. Esterel's `loop P each R` mechanism, for instance, can
 375 be coded using the static parallel composition operator that combines the process `P` to be
 376 aborted and reset with a “watchdog” process `R`. The latter waits for the reset signal from the
 377 environment in each clock cycle. The choice between the reset and continuing inside `P` is
 378 resolved by giving `R` higher priority. When the reset occurs, the watchdog sends “kill” signals
 379 to all threads running in `P` and waits for these to terminate. Only then the watchdog `R`
 380 restarts `P` from scratch. This is a form of precedence that can be expressed in the blocking sets
 381 of CCS^{spt} . As such, Fig. 3 on the left translates the Esterel code of ABRO compositionally in
 382 CCS^{spt} . ABRO is obtained as a parallel composition $(R | ABO) \setminus \{s, t\}$ combining a watchdog
 383 `R` with the behaviour $ABO \stackrel{\text{def}}{=} A | B | O | T$. The `R` is responsible to implement the reset loop at
 384 the superstate called `ABO`, as in the state charts. The `ABO` is the behaviour of the interior
 385 of this superstate, which corresponds to the statement `(await A || await B); emit O` in
 386 Esterel: it is the parallel composition of `A | B` making up the behaviour of the region called
 387 `AB` with code `await A || await B`. The processes `T | O` implement the synchronisation that
 388 waits for termination of `AB` to start the output process `O`. We assume that the channels `a`, `b`,
 389 `r`, `o` modelling interface signals `A`, `B`, `R`, `O` all have a unique sender and receiver. This is seen

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

391 **4** Coherence and Determinacy for CCS^{spt}

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

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

399 Observe that for free processes (i.e., with empty blocking sets) all c-actions are interference-
 400 free and they are c-enabled iff they are admissible. Confluence of interference-free c-actions
 401 is formulated using an auxiliary form of *residual transition* $P \overset{\alpha}{\rightsquigarrow} Q$. This is a transition
 402 from P to Q with an action that is a “factor” of α . Formally, we define $P \overset{\alpha}{\rightsquigarrow} Q$ if (i) $\alpha \in \mathcal{L}$
 403 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 $\ell \in \mathcal{R}$.

404 ▶ **Definition 12** (Process Coherence). *A process P is (structurally) coherent if for all its
 405 derivatives Q the following holds: Given two c-enabled and interference-free transitions
 406 $Q \xrightarrow[E_1]{\alpha_1} Q_1$ and $Q \xrightarrow[E_2]{\alpha_2} Q_2$ such that $\alpha_1 \neq \alpha_2$ or $Q_1 \neq Q_2$ or both $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and
 407 $\alpha_i \notin H_{3-i}$. Then, there exist transitions $Q_1 \xrightarrow[E'_2]{\alpha_2} Q'$ and $Q_2 \xrightarrow[E'_1]{\alpha_1} Q'$ with $H'_i \subseteq H_i$ and
 408 $E_1 \overset{\alpha_2}{\rightsquigarrow} E'_1$ and $E_2 \overset{\alpha_1}{\rightsquigarrow} E'_2$.*

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

410 ▶ **Example 13.** For instance, $Q \stackrel{\text{def}}{=} a:\{ \}.0$ is confluent but not coherent. Since the prefix
 411 not self-blocking, coherence ensures that the label must be *consumable* by arbitrarily many
 412 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
 413 be a transition $Q_i \xrightarrow{a} Q'$ which is not the case. However, for discrete processes, coherence is
 414 identical with confluence: With self-blocking, $Q \stackrel{\text{def}}{=} a:\{a\}.0$ is both confluent and coherent. It
 415 is important to note that all examples in Sec. 3 are coherent but not confluent.

416 ▶ **Theorem 14** (Coherence implies Determinacy). *Every coherent process is determinate under
 417 c-enabled reductions.*

418 A coherent process cannot simultaneously offer both a clock and another distinct action
 419 without putting them in precedence order. Moreover, whenever a clock is enabled, then every
 420 further reduction of the process is blocked by the clock. Thm. 14 concerns the τ -transitions
 421 of a coherent process. For clock transitions we have an even stronger result.

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

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

XX:12 Strong Priority and Determinacy in Timed CCS

429 ▶ **Definition 16** (Precedence Policy). *A precedence policy is a pair $\pi = (L, \dashrightarrow)$ where $L \subseteq \mathcal{L}$*
 430 *is a subset of visible actions, called the alphabet of π and $\dashrightarrow \subseteq L \times L$ a binary relation on*
 431 *L , called the precedence relation.*

432 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
 433 express that the pair (ℓ_1, ℓ_2) is in the precedence relation of the policy. Further, if $\ell_1, \ell_2 \in \pi$
 434 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*
 435 *independent* under π , written (by abuse of the set-notation) $\ell_1 \diamond \ell_2 \in \pi$.

436 For any policy $\pi = (L, \dashrightarrow)$ and set of labels $K \subseteq \mathcal{L}$ we denote by $\pi \setminus K$ the policy
 437 π restricted to the alphabet $L - (K \cup \overline{K})$ in the standard way. There is also the normal
 438 inclusion ordering $\pi_1 \subseteq \pi_2$ between policies defined by $L_1 \subseteq L_2$ and $\dashrightarrow_1 \subseteq \dashrightarrow_2$.

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

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

443 For a process P and policy π , let us write³ $P \Vdash \pi$ to express that P is coherent and conforms
 444 to policy π . We will say that P is *coherent* for π . If $P \Vdash \pi$ then the alphabet of π corresponds
 445 to the sort $\mathcal{L}(P)$ in CCS. It gives an upper bound on the labels that can be used by the
 446 process, and the precedence relation in π provides a static over-approximation of the blocking,
 447 i.e., which labels can be in a choice conflict with each other. As a notational shortcut we will
 448 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,
 449 we write $P \Vdash \ell_1 \diamond \ell_2$ if there is a policy π with $\ell_1 \diamond \ell_2 \in \pi$ and $P \Vdash \pi$.

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

457 ▶ **Example 19.** Reflexive precedences $\ell \dashrightarrow \ell \in \pi$ play a special role in our theory. They
 458 permit a conformant $P \Vdash \pi$ and its derivatives to execute ℓ only with a single partner.
 459 For instance, ABRO from Ex. 10 satisfies $\text{ABRO} \not\vdash r \diamond r$ and so $\text{ABRO} \mid \sigma.\bar{r}$ reduces but
 460 $\text{ABRO} \mid \sigma.\bar{r} \mid \sigma.\bar{r}$ will block after the initial clock transition. In contrast, if the policy specifies
 461 $\ell \diamond \ell \in \pi$, then in a process $P \Vdash \pi$ any ℓ -transition is consumable arbitrarily often. For
 462 instance, the Esterel signal from Ex. 9 has $S_1 \Vdash \text{pres} \diamond \text{pres}$ and so $S_1 \mid \overline{\text{pres}} \mid \overline{\text{pres}}$ will not
 463 block.

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

465 ▶ **Definition 20** (Pivot and precedence-closed policy). *A policy π is called*
 466 *■ pivotable if $\bar{L} \subseteq L$ 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 $\bar{\ell}_1 \diamond \bar{\ell}_2 \in \pi$.*
 467 *■ precedence-closed for $L \subseteq \mathcal{L}$ if $\ell_1 \in L$ and $\ell_1 \dashrightarrow \ell_2 \in \pi$ implies $\ell_2 \in L$.*
 468 *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 CCS^{Ph}.

469 ▶ **Example 21.** An Esterel signal (Ex. 9) has the alphabet $\{emit, pres, abs\} = \mathcal{L}(S_i)$ ($i \in$
 470 $\{1, 2\}$ and for simplicity we ignore the clock σ) and is conformant to the policy psig with
 471 $emit \dashrightarrow abs \in \mathit{psig}$. A typical program R accessing the signal has alphabet $\{\overline{pres}, \overline{abs}\} \subseteq \mathcal{L}(S_i)$
 472 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
 473 is executed, if it is absent then R_0 is executed. Such R conforms to a policy prg with
 474 $\overline{pres} \dashrightarrow \overline{abs} \in \mathit{prg}$ and $\ell \dashrightarrow \ell \in \mathit{prg}$ for $\ell \in \{\overline{pres}, \overline{abs}\}$. Both psig and prg as well as their
 475 union $\mathit{psig} \cup \mathit{prg}$ are pivot policies, so that $R | S_i$ is pivotable. In a pivotable process, conflicting
 476 choices are resolved by the precedences in a single thread, acting as the “pivot”. In $R | S_i$ the
 477 pivot is the signal S_i on reading, resolving the choice $\overline{pres} + \overline{abs}$ in R , and it is the program
 478 R on writing, resolving the choice $abs + emit$ in S_i .

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

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

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

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

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

500 Prop. 23 can be seen as saying that in pivotable processes all clocks take lowest precedence.
 501 As a result, clock transitions satisfy *maximal progress*, i.e., a clock is only enabled on normal
 502 forms, i.e., if there is no reduction possible. Thus clocks behave like time steps in the
 503 standard timed process algebras [18, 11]. More importantly (Thm. 12), all reductions are
 504 confluent. So, a pivotable process, no matter in which order the reductions are executed,
 505 when it terminates, it computes a unique normal form. At this point it either stops if there
 506 is no clock possible, or it offers a clock step (pausing) to a continuation process from which
 507 a new normal form is produced. In this way, coherent pivotable processes correspond to
 508 synchronous streams.

509 It remains to be seen how we can obtain coherent processes systematically by construction.
 510 The following Thm. 24 identifies a set of closure operators for coherent processes based
 511 on policy-conformance. This is our main theorem that summarises Propositions 60–66
 512 (Appendix), which provide in CCS^{spt} a generalisation of Milner’s Confluence Class, Prop. 15–
 513 17 in [27].

XX:14 Strong Priority and Determinacy in Timed CCS

514 ▶ **Theorem 24** (Milner’s Confluence Class Generalised).

515 (1) (Idling) $0 \Vdash \pi$ for all policies π .

516 (2) (Rendezvous Prefix) If $Q \Vdash \pi$, then for every $\ell \in \mathcal{R}$ we have $\ell:H.Q \Vdash \pi$ if $H \subseteq \{\ell' \mid$
517 $\ell' \dashrightarrow \ell \in \pi\}$.

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

519 (4) (Choice) If $P_1 \Vdash \pi_1$ and $P_2 \Vdash \pi_2$ and for all pairs of initial transitions $P \xrightarrow{F_1}_{H_1} P_1$ and
520 $P \xrightarrow{F_2}_{H_2} P_2$ 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
521 $\pi_1 \subseteq \pi$ and $\pi_2 \subseteq \pi$.

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

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

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

525 The theorem is the main result of the paper, namely that under a suitable “policy
526 decoration”, a process can be run deterministically in the scheduled SOS semantics of CCS^{spt} .

527 ▶ **Example 25.** To give a concrete example of the property stated by the above theorem, let’s
528 revisit Ex. 10: the processes A, B, R, O, T are all coherent and conform to the pivot policy
529 π_{ABRO} with precedences (ignoring the clock, so $\pi \setminus \mathcal{R} = \pi$) $k_s \dashrightarrow \bar{t}$, $s \dashrightarrow \bar{t}$, $k_x \dashrightarrow \{k_x, x\}$ for
530 $x \in \{a, b, s, t\}$ and $\ell \dashrightarrow \ell$ for $\ell \in \{r, a, b, \bar{s}, t, \bar{t}, \bar{o}\}$. Thus, by Thm. 24(5), their composition
531 satisfies $A \mid B \mid R \mid O \mid T \Vdash \pi_{\text{ABRO}}$. Since the policy is also precedence-closed for $\{s, t, \bar{s}, \bar{t}\}$, we
532 have $\text{ABRO} \Vdash \pi'_{\text{ABRO}}$ by Thm. 24(3,7), where $\pi'_{\text{ABRO}} = \pi_{\text{ABRO}} \setminus \{s, t\}$.

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

536 ▶ **Definition 26** (Milner’s Confluence Class). *Milner’s confluence class $\text{CCS}^{\text{cc}} \subseteq \mathcal{P}$ is the set*
537 *of processes P such that for all derivatives Q of P , if $Q \xrightarrow{\alpha}_E Q'$, the following holds:*

538 ■ *If $\alpha \neq \tau$ then $H = \{\alpha\}$ and $\alpha \notin \bar{iA}^*(E)$.*

539 ■ *If $\alpha = \ell \mid \bar{\ell}$ then $H = \{\}$ and $\{\ell, \bar{\ell}\} \cap \bar{iA}^*(E) = \{\}$.*

540 ▶ **Conjecture 1** (Milner’s Confluence Class is Coherent). *For all $P, Q \in \text{CCS}^{\text{cc}}$:*

541 ■ *P is coherent iff P is confluent.*

542 ■ *A transition of P is c-enabled iff it is admissible.*

543 ■ *If $\mathcal{L}(P) \cap \mathcal{L}(Q) = \{\}$ and $\overline{\mathcal{L}(P)} \cap \mathcal{L}(Q) = L \cup \bar{L}$, then $(P \mid Q) \setminus L \in \text{CCS}^{\text{cc}}$.*

544 **5 Further work**

545 We list in spare order some possible future works:

546 ■ We plan to study the interaction of our notion of coherence with forms of congruence
547 (static, dynamic and interaction laws in Milner’s jargon) and notions of bisimulation.

548 ■ We plan to generalise the interaction model of CCS^{spt} to synchronisation algebras
549 permitting action fusion in the spirit of Milner’s ASCCS and Austry and Boudol’s
550 Meije (see [26, 3]).

551 ■ Thm. 24 suggests using policies as types. Among possible type disciplines we just
552 mention: behavioural type systems, including session-types to capture rendezvous
553 interactions; intersection-types, including non-idempotent ones to characterise termination
554 and resources awareness, and linear-types to model consuming resources.

- 555 ■ C-actions are somewhat related with continuation passing style, and, in some sense they
556 are more powerful because they permit a “forecast” analysis of all possible future actions:
557 studying this relation, possibly in relation with abstract interpretation techniques, would
558 be worth of study.
- 559 ■ Clocks are unordered in CCS^{spt} : introducing a partial order could be an interesting,
560 although economic and fine grained, way to reason about time and time priorities.



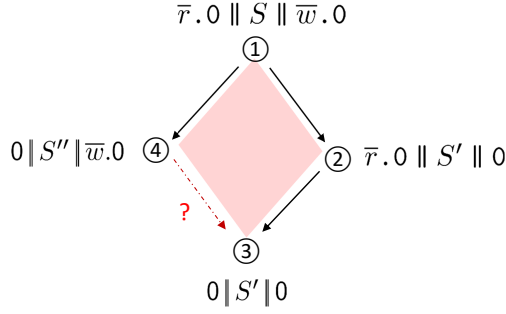
561 — **References** —

- 562 1 J. Aguado, M. Mendler, R. v. Hanxleden, and I. Fuhrmann. Grounding synchronous
563 deterministic concurrency in sequential programming. In Proceedings of the 23rd European
564 Symposium on Programming (ESOP'14), LNCS 8410, pages 229–248, Grenoble, France, April
565 2014. Springer.
- 566 2 R. M. Amadio. A synchronous pi-calculus. Inf. Comput., 205(9):1470–1490, 2007. URL:
567 <https://doi.org/10.1016/j.ic.2007.02.002>, doi:10.1016/J.IC.2007.02.002.
- 568 3 D. Austry and G. Boudol. Algèbre de processus et synchronisation. Theor. Comput. Sci.,
569 30:91–131, 1984.
- 570 4 G. Barrett. The semantics of priority and fairness in OCCAM. In M. Main, A. Melton,
571 M. Mislove, and D. Schmidt, editors, Proc. MFPS'89, 1989.
- 572 5 G. Berry. The Constructive Semantics of Pure Esterel. Draft Book, 1999. [ftp://ftp-sop.](ftp://ftp-sop.inria.fr/esterel/pub/papers/constructiveness3.ps)
573 [inria.fr/esterel/pub/papers/constructiveness3.ps](ftp://ftp-sop.inria.fr/esterel/pub/papers/constructiveness3.ps).
- 574 6 G. Berry. The Foundations of Esterel. In Proof, Language and Interaction: Essays in Honour
575 of Robin Milner, pages 425–454. MIT Press, 2000.
- 576 7 G. Berry and G. Boudol. The chemical abstract machine. Theor. Comput. Sci.,
577 96(1):217–248, 1992. URL: [https://doi.org/10.1016/0304-3975\(92\)90185-I](https://doi.org/10.1016/0304-3975(92)90185-I), doi:10.
578 1016/0304-3975(92)90185-I.
- 579 8 R. Bocchino, V. Adve, S. Adve, and M. Snir. Parallel programming must be deterministic by
580 default. In Proceedings of the First USENIX conference on Hot topics in parallelism, pages
581 4–4, 2009.
- 582 9 F. Boussinot and R. De Simone. The SL synchronous language. IEEE Transactions on
583 Software Engineering, 22(4):256–266, 1996.
- 584 10 J. Camilleri and G. Winskel. CCS with priority choice. Information and Computation,
585 116(1):26–37, 1995.
- 586 11 R. Cleaveland, G. Lüttgen, and M. Mendler. An algebraic theory of multiple clocks. In
587 A. Mazurkiewicz and J. Winkowski, editors, Proceedings of the International Conference on
588 Concurrency Theory CONCUR'97, pages 166–180. Springer, 1997. LNCS 1243.
- 589 12 V. Dieckert and G. Rozenberg, editors. The Book of Traces. World Scientific, 1995.
- 590 13 S. A. Edwards. On determinism. In M. Lohstroh et. al., editor, Lee Festschrift, pages 240–253.
591 Springer, 2018.
- 592 14 P. Le Guernic, T. Goutier, M. Le Borgne, and C. Le Maire. Programming real time applications
593 with SIGNAL. In Proceedings of the IEEE, volume 79, pages 1321–1336. IEEE Press,
594 September 1991.
- 595 15 M. Mendler H. R. Andersen. An asynchronous process algebra with multiple clocks. In
596 D. Sannella, editor, In Proc. ESOP'94, pages 58–73, 1994.
- 597 16 N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming
598 language LUSTRE. Proceedings of the IEEE, 79(9):1305–1320, September 1991.
- 599 17 D. Harel, A. Pnueli, J. Pruzan-Schmidt, and R. Sherman. On the formal semantics of
600 Statecharts. In Logic in Computer Science (LICS'87), pages 54–64. IEEE Press, 1987.
- 601 18 M. Hennessy and T. Regan. A process algebra for timed system. Information and Computation,
602 117:221–239, 1995.
- 603 19 C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985. 256 pages.
- 604 20 C. D. Kloos and P. Breuer, editors. Formal Semantics for VHDL. Kluwer, 1995.
- 605 21 J. W. Klop. Combinatory reduction systems. PhD thesis, Univ. Utrecht, 1980.
- 606 22 L. Kuper and R. Newton. Lvars: lattice-based data structures for deterministic parallelism.
607 In Functional High-Performance Computing, page 71–84, New York, NY, USA, 2013. ACM.
- 608 23 E. A. Lee. The problem with threads. Computer, 39(5):33–42, 2006.
- 609 24 G. Lüttgen, M. von der Beeck, and R. Cleaveland. Statecharts via process algebra. In
610 CONCUR '99: Concurrency Theory, volume 1664 of Lecture Notes in Computer Science,
611 pages 399–414. Springer, 1999.

- 612 25 S. Marlow. Parallel and concurrent programming in haskell. In *CEFP 2011*, pages 339–401,
613 2012.
- 614 26 R. Milner. Calculi for synchrony and asynchrony. *Theor. Comput. Sci.*, 25:267–310, 1983. URL:
615 [https://doi.org/10.1016/0304-3975\(83\)90114-7](https://doi.org/10.1016/0304-3975(83)90114-7), doi:10.1016/0304-3975(83)90114-7.
- 616 27 R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- 617 28 R. Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press,
618 1999.
- 619 29 R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*,
620 100(1):1–40, 1992. URL: [https://doi.org/10.1016/0890-5401\(92\)90008-4](https://doi.org/10.1016/0890-5401(92)90008-4), doi:10.1016/
621 0890-5401(92)90008-4.
- 622 30 R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, II. *Inf. Comput.*,
623 100(1):41–77, 1992. URL: [https://doi.org/10.1016/0890-5401\(92\)90009-5](https://doi.org/10.1016/0890-5401(92)90009-5), doi:10.1016/
624 0890-5401(92)90009-5.
- 625 31 X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: theory and application. *Inf. Comput.*,
626 114(1):131–178, 1994. URL: <https://doi.org/10.1006/inco.1994.1083>, doi:
627 10.1006/INCO.1994.1083.
- 628 32 B. Norton, G. Luetngen, and M. Mendler. A compositional semantic theory for component-
629 based synchronous design. In R. Amadio and D. Lugiez, editors, *Int'l Conf. on Concurrency*
630 *Theory (CONCUR'2003)*, pages 461–476, Marseille, September 2003. Springer LNCS 2761.
- 631 33 I. Phillips. CCS with priority guards. In K. G. Larsen and M. Nielsen, editors,
632 *Proc. Concur 2001*, pages 305–320, 2001.
- 633 34 G. D. Plotkin. The origins of structural operational semantics. *J. Log. Algebraic Methods*
634 *Program.*, 60-61:3–15, 2004. URL: <https://doi.org/10.1016/j.jlap.2004.03.009>, doi:
635 10.1016/J.JLAP.2004.03.009.
- 636 35 G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebraic Methods*
637 *Program.*, 60-61:17–139, 2004.
- 638 36 V. Natarajan R. Cleaveland, G. Luetngen. *Handbook of Process Agebra*, chapter Priority in
639 Process Algebra. Elsevier, 2001.
- 640 37 K. Schneider. The synchronous programming language Quartz. Internal Report 375,
641 Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany,
642 December 2009.
- 643 38 Esterel Technologies. The Esterel v7 reference manual version v7_30 – initial IEEE
644 standardization proposal. Technical report, Esterel Technologies, 679 av. Dr. J. Lefebvre,
645 06270 Villeneuve-Loubet, France, November 2005.
- 646 39 S.H. Unger. Asynchronous sequential switching circuits with unrestricted input changes. *IEEE*
647 *Transactions on Computers*, C-20(12):1437–1444, 1971. doi:10.1109/T-C.1971.223155.
- 648 40 L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111,
649 1990. URL: <https://doi.org/10.1145/79173.79181>, doi:10.1145/79173.79181.
- 650 41 R. von Hanxleden, B. Duderstadt, C. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, and
651 O. O'Brien. SCCharts: Sequentially Constructive Statecharts for safety-critical applications.
652 In *Proc. PLDI'14*, Edinburgh, UK, June 2014. ACM.

653 **A** From Asynchronous Priorities to Synchronous ones, via Clocks

654 This rather informal section gives to the curious reader a nice view of our personal pilgrimage
655 of previous work looking at reaching more control and determinism in asynchronous process
656 algebra. This trip lead us to consider adding clocks to the previous literature and reach
657 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.

658 **A.1 CCS with (Weak) Priorities**

659 Let us look in more detail on how a precedence mechanism such as “write-before-read” might
 660 be derived from the classical priorities studied for CCS. As a simple example, suppose S is a
 661 (synchronous write, asynchronous read) memory cell that can be read at any time offering
 662 the read action r , but only be written once with the write action w . Ignoring data values,
 663 the cell would be modelled in plain CCS by the equations $S \stackrel{def}{=} w.S' + r.S''$ where $S' \stackrel{def}{=} r.S'$
 664 and $S'' \stackrel{def}{=} r.S'' + e.S''$. Initially, memory cell S is empty and offers a choice between a read
 665 action r and a write action w . If the write action is performed, the cell receives a value and
 666 changes to state S' . In this filled state S' , only read actions are admissible to access the
 667 stored value, the state does not change. When the read action is executed on the empty cell
 668 S , however, the continuation state is S'' which represents an error state. In the error state
 669 S'' only a read or an error action e is possible, but no write and the cell remains in error.
 670 Observe that only the read action is always admissible in both states S and S' , while the
 671 write is admissible only in state S . If S is put in parallel with a reader offering \bar{r} and a writer
 672 offering \bar{w} , then the final state of S depends on the order between the \bar{r} and \bar{w} actions. If we
 673 synchronise with the external sequence $\bar{r}.\bar{w}$ on S where the read action is first, the writer
 674 gets blocked on the error state S'' , while if we exercise $\bar{w}.\bar{r}$ where the write action is first,
 675 both can proceed and we obtain filled state S' . In term rewriting system jargon [21] this is
 676 called “critical pairs”.

677 Formally, let $R \stackrel{def}{=} \bar{r}.0$ be a reader process with a single read and $W \stackrel{def}{=} \bar{w}.0$ the writer
 678 with a single write, both terminating in the empty behaviour 0 . Then, according to the
 679 operational semantics of CCS, we have the two execution sequences $R \mid S \mid W \xrightarrow{\tau} 0 \mid S'' \mid W$
 680 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 $0 \mid S' \mid 0$
 681 are distinct normal forms, in the sense that no reductions are possible anymore. This is pictorially
 682 illustrated in Fig. 4. Observe that the behaviour is not only *non-deterministic* in the sense
 683 that there is more than one scheduling sequence. It is also non-determinate as the final
 684 outcome depends on the scheduling decisions.

685 Among many CCS extension with priorities, we looked the one of Camilleri and Winskell [10],
 686 named CCS^{CW} in the sequel. CCS^{CW} provides a *prioritised choice* operator $Q \stackrel{def}{=} \alpha_1.Q_1 \dot{\rightarrow}$
 687 $\alpha_2.Q_2$ called “prisum” that gives the action α_1 priority over α_2 and blocks the second prefix α_2
 688 in case $\alpha_1 = \alpha_2$. Such priorities help us resolve the issue. The problem with non-determinism
 689 originates from the preemptive behaviour of the write-read choice $w.S' + r.S''$. It permits
 690 the read and the write actions to compete with each other for the next state of the process

691 S . In data-flow and functional programming, we resolve the conflict by making the write
 692 action take precedence over the read one. First, the data-flow variable must be written by
 693 the *producer* process, then it can be read by the *consumers*, if any. This “write-before-read”
 694 scheduling policy is a *prioritised choice*, represented in CCS^{CW} as $S \stackrel{\text{def}}{=} w.S' \dot{\rightarrow} r.S''$ in which
 695 the prism $\dot{\rightarrow}$ operator enforces a precedence from left to right. The store S is willing to
 696 engage in a read action r (to the right of $\dot{\rightarrow}$) only if the environment does not offer a write w
 697 (on the left of $\dot{\rightarrow}$) at the same time, concurrently.

698 Another important study in the endeavour of adding priorities to CCS is the work of
 699 Phillips [33]. In Phillips’ extension of CCS with priority guards, referred to as CCS^{Ph} , the
 700 previous behavior is expressed by $S \stackrel{\text{def}}{=} \{ \} : w . S' + \{ w \} : r . S''$, where each action is pre-annotated
 701 by a set of actions that take priority. Process S will synchronise with the reader process R
 702 to move to the error state S'' with *guarded action* $\{ w \} : r$. The *priority guard* $\{ w \}$ states that
 703 this can only happen if the environment does not offer a write w . Otherwise, the only action
 704 available is $\{ \} : w$ which makes the cell move into state S' . The write action has an empty
 705 guard and so cannot be blocked. If both \bar{w} and \bar{r} are offered concurrently, then only the
 706 higher-prioritised write action \bar{w} will go ahead.

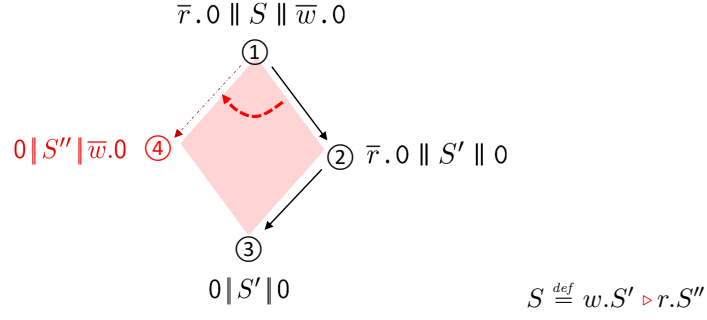
707 The standard way to implement this scheduling in prioritised CCS is via a transition
 708 relation that keeps track of blocking information. In line with the approaches of [10, 33, 36],
 709 let us write $P \xrightarrow{\alpha}_H P'$ to denote that process P can engage in an action α to continue as P' ,
 710 provided the concurrent environment does not offer any action from H . We will call H the
 711 *blocking set* of the transition. In Plotkin’s SOS, the (selected) rules for prefix and choice in
 712 CCS^{CW} are the following⁴:

$$713 \quad \frac{}{\alpha.P \xrightarrow{\alpha}_{\{ \}} P} (\text{Act}) \quad \frac{P \xrightarrow{\alpha}_H P'}{P \dot{\rightarrow} Q \xrightarrow{\alpha}_L P'} (\text{PriSum}_1) \quad \frac{Q \xrightarrow{\alpha}_H Q' \quad \tau, \alpha \notin \text{iA}(P)}{P \dot{\rightarrow} Q \xrightarrow{\alpha}_{H \cup \text{iA}(P)} Q'} (\text{PriSum}_2)$$

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

$$726 \quad \frac{P_1 \xrightarrow{\alpha_1}_{H_1} P'_1 \quad H_1 \cap \overline{\text{iA}}(P_2) = \{ \}}{P_1 | P_2 \xrightarrow{\alpha_1}_H P'_1 | P_2} (\text{Par}_1) \quad \frac{P_2 \xrightarrow{\alpha_2}_{H_2} P'_2 \quad H_2 \cap \overline{\text{iA}}(P_1) = \{ \}}{P_1 | P_2 \xrightarrow{\alpha_2}_{H_2} P_1 | P'_2} (\text{Par}_2)$$

⁴ For the fragment of CCS^{CW} that we are considering, the precise connection is the following: $\vdash_R P \xrightarrow{\alpha} P'$
 in CCS^{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 $\bar{\alpha} \in R$ and $R \cap \bar{H} = \{ \}$.



■ **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.

$$\begin{array}{c}
 727 \\
 \frac{P_1 \xrightarrow{\alpha_1}_{H_1} P'_1 \quad P_2 \xrightarrow{\alpha_2}_{H_2} P'_2 \quad \alpha_1 = \bar{\alpha}_2 \quad H_1 \cap \bar{iA}(P_2) = \{\} \quad H_2 \cap \bar{iA}(P_1) = \{\}}{P_1 | P_2 \xrightarrow{\tau}_{H_1 \cup H_2} P'_1 | P'_2} \quad (Par_3) \\
 728
 \end{array}$$

729 More in details, rules (Par_i) for $i \in \{1, 2\}$ define an action α_i by $P_1 | P_2$ offered in one of the
 730 sub-processes P_i . Such is enabled if the competing concurrent process P_{3-i} has no initial
 731 action that is in the blocking set H_i of action α_i , i.e. when the condition $H_i \cap \bar{iA}(P_{3-i}) = \{\}$
 732 is satisfied. Rule (Par_3) implements the synchronisation between an action α_1 in P_1 and the
 733 associated matching action $\alpha_2 = \bar{\alpha}_1$, originating from P_2 . The side-condition is the same:
 734 each process P_i only accepts the matching action call from P_{3-i} if it does not offer any action
 735 that blocks action α_i . Let us see how this blocking semantics works for our running example
 736 of a write-once memory cell, where we now enforce a write-before-read priority on the empty
 737 cell S : $S \stackrel{def}{=} w.S' \dot{\triangleright} r.S''$ and $S' \stackrel{def}{=} r.S'$ and $S'' \stackrel{def}{=} r.S'' + e.S''$. The cell S has two potential
 738 transitions that can be fired, namely

$$\begin{array}{c}
 \frac{w.S' \xrightarrow{w}_{\{\}} S'}{S \xrightarrow{w}_{\{\}} S'} \quad (PriSum_1) \quad \text{or} \quad \frac{r.S'' \xrightarrow{r}_{\{\}} S'' \quad \tau, r \notin iA(w.S')}{S \xrightarrow{r}_{\{w\}} S''} \quad (PriSum_2) \\
 739
 \end{array}$$

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

$$\begin{array}{c}
 \frac{\frac{R \xrightarrow{\bar{r}}_{\{\}} 0 \quad (Act) \quad \frac{\vdots}{S \xrightarrow{r}_{\{w\}} S''} \quad \{\} \cap \bar{iA}(S) = \{\} \quad \{w\} \cap \bar{iA}(R) = \{\}}{R | S \xrightarrow{\tau}_{\{w\}} 0 | S''} \quad (Par_3)}{742}
 \end{array}$$

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

749 The derivation tree

$$\begin{array}{c}
 \frac{\frac{\frac{R \xrightarrow{\tau} \{ \} 0}{(Act)} \quad \frac{\frac{r.S'' \xrightarrow{\tau} \{ \} S''}{(Act)} \quad \frac{S \xrightarrow{\tau} \{w\} S''}{(PriSum_2)}}{R | S \xrightarrow{\tau} \{w\} 0 | S''} (Par_3) \quad \{w\} \cap iA(W) \neq \{ \}}{R | S | W \xrightarrow{\tau} \{w\} 0 | S'' | W} (Par_1)
 \end{array}$$

750

751 where the side-conditions of $(PriSum_2)$ and of (Par_3) are satisfied, blocks at side-condition
 752 of rule (Par_1) in red. On the other hand, the writing of S by W followed by the reading of
 753 S' by R can be derived, i.e., we have $R | S | W \xrightarrow{\tau} \{ \} R | S' | 0$ and $R | S' | 0 \xrightarrow{\tau} \{ \} 0 | S' | 0$. The
 754 derivation trees in the SOS are as follows

$$\begin{array}{c}
 \frac{\frac{\frac{w.S' \xrightarrow{w} \{ \} S'}{(Act)} \quad \frac{S \xrightarrow{w} \{ \} S'}{(PriSum_1)}}{R | S \xrightarrow{w} \{ \} R | S'} (Par_2) \quad \frac{W \xrightarrow{w} \{ \} 0}{(Act)} \quad \frac{R \xrightarrow{\tau} \{ \} 0}{(Act)} \quad \frac{\frac{S' \xrightarrow{\tau} \{ \} S'}{(Act)} \quad \frac{S' | 0 \xrightarrow{\tau} \{ \} S' | 0}{(Par_3)}}{R | S' | 0 \xrightarrow{\tau} \{ \} 0 | S' | 0} (Par_3)}{R | S | W \xrightarrow{\tau} \{ \} R | S' | 0} (Par_3)
 \end{array}$$

755

756 where the blocking side-conditions of (Par_1) , (Par_2) and (Par_3) are all trivially satisfied.
 757 The scheduling which resolves the non-determinism is depicted in Fig. 5.

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

759 Our example from the previous section shows how (weak) priority-based scheduling helps
 760 us to exercise control over the execution order in an asynchronous process calculus with
 761 rendez-vous communication. The surprising observation is that very little needs to be
 762 changed in the traditional notion of priority for CCS to achieve confluence. For this we
 763 need to look how to “tune” of the classical weak priorities for our purposes. Our fix will
 764 make the difference between (*weak*) *priority-based* and (*strong*) *priority-based* scheduling (see
 765 Def. 3). What we want is that every process using only strong priority satisfies the following
 766 *weak local confluence* property [21]. Suppose P admits two transitions to different states
 767 $P_1 \neq P_2$ 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 $\alpha_1 \notin H_2$
 768 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
 769 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
 770 to express non-interference between transitions. Then, our weak⁵ local confluence expresses
 771 a weak “diamond property” stating that every diverging choice of non-interfering transitions
 772 immediately reconverges. The subset inclusions $H'_i \subseteq H_i$ ensure that delaying an admissible
 773 action does not introduce more chances of interference.

774 A.2.1 Fix 1

775 The standard notion of priority for CCS, as implemented in CCS^{CW} or CCS^{Ph} , of course was
 776 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.

XX:22 Strong Priority and Determinacy in Timed CCS

777 express scheduling control. Even in the fragment of CCS^{CW} where all choices are prioritised,
778 weak confluence fails. For instance, although our memory cell S behaves deterministically
779 for a single reader R and a single writer W , in the context $E \stackrel{\text{def}}{=} W_1 | [\cdot] | W_2$ of two writers
780 $W_i \stackrel{\text{def}}{=} \bar{w}.W'_i$, it creates a race. The two writers compete for the single write action on S
781 and only one can win. Using the operational semantics of CCS^{CW} we can derive the (non-
782 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,
783 the two outcomes of the divergence are a critical pair that have no reason to reconverge
784 $W'_1 | S' | W_2 \rightarrow P'$ and $W_1 | S' | W'_2 \rightarrow P'$ if W'_i are arbitrary behaviours. To preserve confluence
785 we must block the environment from consuming the single prefix $w.S'$ twice. The problem is
786 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
787 sets $H_i = \{ \}$ which does not prevent the second writer W_{3-i} to be added in parallel. If
788 $w \in H_i$, then the rules (Par_{3-i}) would automatically prevent the second reader. Now, suppose
789 we extend the action rule (Act) to expose the singleton nature of a prefix, like by (Act^+),
790 $\alpha.P \xrightarrow{\alpha}_{\{ \alpha \}} P$ we would not only prevent a second concurrent writer but already a single
791 writer. The transitions $W_1 \xrightarrow{\bar{w}}_{\{ \bar{w} \}} W'_1$ and $S \xrightarrow{w}_{\{ w \}} S'$ would not be able to communicate,
792 violating both side-condition $\{ \bar{w} \} \cap \bar{iA}(S) = \{ \}$ and $\{ w \} \cap \bar{iA}(W_1) = \{ \}$ of (Par_3). The
793 issue is that in the setting of CCS^{CW} (and CCS^{Ph} , indeed) the blocking set H of a transition
794 $P \xrightarrow{\alpha}_H P'$ expresses a priority rather than an interference. An action cannot take priority
795 over itself without blocking itself. Therefore, in CCS^{CW} (and CCS^{Ph}) we always have $\alpha \notin H$.
796 Here, we need to apply a weaker interpretation. The actions in H only prevent α from
797 synchronising with a matching $\bar{\alpha}$ in the environment, if they are executed in a *different* (and
798 thus competing) prefix from the one that executes $\bar{\alpha}$. Actions in H *interfere* with α in the
799 sense that they are initial actions of P sharing the same control thread in which α is offered.
800 Thus, we want to avoid a situation in which the environment synchronises with $\bar{\alpha}$ in the
801 presence of *another distinct* communication on some $\beta \in H$. For then we would have a race
802 for the behaviour of the thread shared by α and β in P .

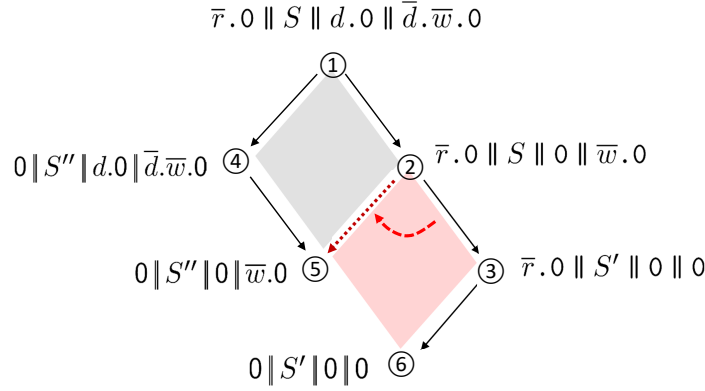
803 To fix this, we need to refine the side-conditions $H_i \cap \bar{iA}(P_{3-i})$ in (Par_3) to check
804 not *all* initial actions $\bar{iA}(P_{3-i})$ but only those initial actions of P_{3-i} that are *competing*
805 with the matching transitions $P_1 \xrightarrow{\alpha_1}_{H_1} P'_1$ and $P_2 \xrightarrow{\alpha_2}_{H_2} P'_2$ with $\alpha_1 = \bar{\alpha}_2$, involved in the
806 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
807 information exposed by a transition. More specifically, we extend each transition $P \xrightarrow{\alpha}_K P'$
808 by the set K of initial actions that are offered in competing prefixes of the same thread as α
809 or by threads concurrent to the thread from which α is taken. Then, the set of all initial
810 actions is simply $iA(P) = K \cup \{ \alpha \}$. Let us call K the *initial environment* of the transition.
811 From the description, we expect the invariant that $H \subseteq K$.

812 ▶ Remark 27. Note that we cannot simply define $K \stackrel{\text{def}}{=} iA(P) - \{ \alpha \}$, because α could be
813 contained in K . For example, take $P \stackrel{\text{def}}{=} \alpha.0 | \alpha.0$, where $P \xrightarrow{\alpha}_{\{ \alpha \}} \{ \alpha \}.0$. The blocking set is
814 $H = \{ \alpha \}$ since the prefix is volatile and must be protected from a competing consumptions.
815 Also, we need α to be in the initial environment $K = \{ \alpha \}$, because the transition's action α
816 is offered a second time in a concurrent thread. This is different from the set $iA(P) - \{ \alpha \} =$
817 $\{ \alpha \} - \{ \alpha \} = \{ \}$. This shows that we must compute the initial environment explicitly as an
818 extra label with each transition.

819 Now we can reformulate the parallel rule as

$$\frac{P_i \xrightarrow{\alpha_i}_{K_i} P'_i \quad \alpha_1 = \bar{\alpha}_2 \quad H_i \cap \bar{K}_{3-i} = \{ \} \quad \text{for } i \in \{1, 2\}}{P_1 | P_2 \xrightarrow{\tau}_{K_1 \cup K_2} P'_1 | P'_2} (\text{Par}_3^+)$$

820



■ **Figure 6** The store S in composition with a reader $\bar{r}.0$ and a delayed writer $d.0 \mid \bar{d}.\bar{w}.0$. 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 transitions are silent with labels omitted.

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

823 A.2.2 Fix 2

824 In the previous subsection we have seen how priorities can be beneficial to approaching to
825 a deterministic scheduling in an intrinsic non-deterministic process algebra. The priority
826 blocking eliminates the competition between mutually pre-empting actions and the remaining
827 independent actions naturally commute. Now another question naturally arises: “does
828 this mean that if we replace the non-deterministic choice $P + Q$ by a prioritised choice,
829 denoted by $P \dot{\vdash} Q$, do we obtain a determinate process algebra?” Indeed, we can show
830 that in the “asynchronous” fragment of CCS, i.e. where all prefixes $\alpha.P$ are of form $\alpha.0$,
831 confluence can be granted: On the contrary, in the “synchronous” fragment of CCS, where
832 actions sequentially guard new actions, we run into a problem of non-monotonicity. A
833 communication between prefixes may trigger new actions that block another communication
834 that was enabled before. In other words, it is not guaranteed that independent actions remain
835 admissible. As an example, in the above reader-writer scenario, the actions a_i ($i \in \{1, 2\}$)
836 may become blocked in processes P_{3-i} . Now, consider a parallel composition $R \mid S^* \mid W^*$
837 in which a modified store $S^* \stackrel{def}{=} S \mid d.0$ offers another concurrent method action d on the
838 side in parallel. Let the writer W^* be delayed by a synchronisation with this concurrent
839 action d , i.e., $W^* \stackrel{def}{=} \bar{d}.W \stackrel{def}{=} \bar{d}.\bar{w}.0$. The scheduling sequences are depicted in Fig. 6. We can
840 still derive $S \mid d.0 \xrightarrow[r]{\{w,d\}}_{\{r,w\}} S'' \mid d.0$ but also $S \mid d.0 \xrightarrow[d]{\{r,w\}}_{\{d\}} S$. The reading synchronisation
841 $R \mid S \mid d.0 \xrightarrow[\{w,d\}]{\tau}_{\{\bar{r},r,w\}} S'' \mid d.0$ can now take place by (Par_3) before the write to give the
842 following reduction sequence: $R \mid S \mid d.0 \mid W^* \xrightarrow[\{w,d,\bar{d}\}]{\tau}_{\{\bar{r},r,w\}} S'' \mid d.0 \mid W^* \xrightarrow[\{r,e\}]{\tau}_{\{d\}} S'' \mid W$. The
843 first reduction is justified by rule (Par_1), because the side-condition $\{w\} \cap \bar{i}\bar{A}(W) = \{w\} \cap \{d\} =$
844 $\{\}$ is satisfied. The blocking side-condition which only looks at the moment when the read is
845 happening does not see the write action \bar{w} by W^* hidden behind its initial action \bar{d} . It lets
846 process R access and read the store S . As before, however, the following reduction sequence is
847 allowed, too: $R \mid S \mid d.0 \mid W^* \xrightarrow[\{\bar{r},r,w\}]{\tau}_{\{d\}} R \mid S \mid W \xrightarrow[\{\bar{r},r\}]{\tau}_{\{w,\bar{w}\}} R \mid S' \xrightarrow[\{w\}]{\tau}_{\{\bar{r},r,w\}} S'$ in which the

848 write action happens before the read action. This creates two diverging reduction sequences
 849 (incoherent schedules) in which the final store is S' or S'' , non-deterministically.

850 Our insight is that the blocking side-conditions of the standard Camilleri-Winskill
 851 theory [10], described above, are too weak in scheduling control for our purposes and to
 852 guaranteeing a weak Church-Rosser property [21]. The problem is that in the side-condition
 853 of rule Par_2 we are only looking at the initial actions of Q . This is why the read action is
 854 blocked out of state ② in Fig. 6 but not out of state ①. In state ① there is no pending initial
 855 write. The write action \bar{w} only becomes initial when the parallel processes $\bar{d}.0 \mid d.\bar{w}.0$ have
 856 interacted on action d and advanced together with a silent action τ , i.e. $\bar{d}.0 \mid d.\bar{w}.0 \xrightarrow{\tau} 0 \mid \bar{w}.0$.
 857 In order to see that the read action r blocked by $\{w\}$ out of state ① should not be enabled,
 858 we must follow all sequences of interactions that can possibly happen concurrently to the
 859 read. The read is to be blocked, because after one silent τ -step, in the reachable configuration
 860 of state ②, the write action w in the blocking set $\{w\}$ of a silent action, meets a matching
 861 initial partner \bar{w} .

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

$$873 \frac{P \xrightarrow[E]{a} P' \quad E' = E \mid Q \quad H \cap \overline{iA^\tau}(E') = \{\}}{P \mid Q \xrightarrow[E']{a} P' \mid Q} (Par_1^*)$$

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

$$876 \frac{P \xrightarrow[E_1]{a} P' \quad Q \xrightarrow[E_2]{\bar{\alpha}} Q' \quad E = E_1 \mid E_2 \quad \forall i. H_i \cap (\overline{iA^\tau}(E) \cup L_{3-i}) = \{\}}{P \mid Q \xrightarrow[E]{\tau} P' \mid Q'} (Par_3^*)$$

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

$$878 R \xrightarrow[0]{\bar{r}} R' \text{ and } S \mid d.0 \xrightarrow[d.0]{r} S'', \text{ we get } R \mid S \mid d.0 \xrightarrow[d.0]{\tau} S'' \mid d.0.$$

879 The side-condition of (Par_3^*) is satisfied since the “weak” initial action set (where τ actions are
 880 dropped) $iA^\tau(d.0) = \{d\}$. Thus, both $\{r\} \cap \overline{iA^\tau}(d.0) = \{\}$ as well as $\{w, r\} \cap \overline{iA^\tau}(d.0) = \{\}$.
 881 If now want to shunt this past the writer process W^* using rule (Par_1^*), then we get
 882 stuck. Since $d.0 \mid W^* \xrightarrow[0]{\tau} W$ and $\bar{w} \in iA(W)$, we find $\bar{w} \in iA^\tau(d.0 \mid W^*)$ and thus
 883 $\{w\} \cap \overline{iA^\tau}(d.0 \mid W^*) \neq \{\}$. Therefore, the side-condition of (Par_1^*) is violated and the
 884 reading interaction of $R \mid S \mid d.0$ cannot be composed with W^* . This is what we want.
 885 The transition from state ② to state ③ in Fig. 6 is now blocked. On the other hand,
 886 the reduction $R \mid S \mid d.0 \xrightarrow[R \mid S]{d} R \mid S$ can be composed with $W^* \xrightarrow[0]{\bar{d}} W$ via (Par_3^*)
 887 to make a reduction $R \mid S \mid d.0 \mid W^* \xrightarrow[R \mid S]{\tau} R \mid S \mid W$. Regarding the side-condition, we

888 notice that the weak initial action set $iA^\tau(R|S)$ of the reduction context is irrelevant, because
 889 it does not contain d or \bar{d} . In configuration $R|S|W$ (state ② in Fig. 6), the reader R cannot
 890 interact with S . The associated reduction $R|S|W \xrightarrow[\bar{W}]{\tau}_{\{r,w\} \cup \{\}} S''|W$ is blocked by the
 891 reduction environment, because $w \in \bar{iA}^\tau(\bar{w}.W')$. For our further discussion, let us call the
 892 proposed change in the side-conditions leading to (Par_i^*) from (Par_i^*) a process algebra with
 893 *strong priorities*⁶ and let's call the traditional process algebra with priorities, introduced by
 894 [10, 36], as process algebra with *weak priorities*.

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

898 ▶ **Definition 28** (Weak Initial Actions & Strong Enabling). *The set $iA^\tau(P)$ of weak initial*
 899 *transitions is the smallest extension $iA(P) \subseteq iA^\tau(P)$ such that if $\alpha \in iA^\tau(Q)$ and $P \xrightarrow{\tau} Q$*
 900 *then $\alpha \in iA^\tau(P)$. Let $\bar{iA}^\tau(P) = \bar{iA}^\tau(\bar{P})$. A transition $P \xrightarrow[\bar{R}]{\alpha}_H P'$ is called *strongly enabled* if
 901 $H \cap (\bar{iA}^\tau(R) \cup \{\tau\}) = \{\}$.*

902 It is easy to see that $iA(P) \subseteq iA^\tau(P)$, so every strongly enabled transition is weakly enabled:
 903 as such, for free processes, the notions of weak and strong enabling coincide. Note also
 904 that enabling as a scheduling constraint is not monotonic under parallel composition. The
 905 enabledness property of a transition is not, in general, preserved when the process is placed into
 906 a concurrent context. If a reduction $P \xrightarrow[\bar{R}]{\tau}_H P'$ is enabled, then we know that $H \cap \bar{iA}(R) = \{\}$
 907 or $H \cap \bar{iA}^\tau(R) = \{\}$, depending on what scheduling we choose. This means that the context R
 908 cannot (weakly) generate a synchronisation with a blocking initial action. This does not mean
 909 that an extension $R|Q$ may not perhaps have $H \cap \bar{iA}(R|Q) \neq \{\}$ or $H \cap \bar{iA}^\tau(R|Q) \neq \{\}$.
 910 This happens if blocking actions come from Q or, in the case of strong enabling, become
 911 reachable though the interactions of R and Q . Then the parallel extension $P|Q$ does not
 912 permit a reduction $P|Q \xrightarrow[\bar{R}]{\tau}_H P'|Q$. As indicated by the examples of Sec. 3, strong enabling
 913 generates confluent reductions in the reflexive case, where rendez-vous actions have a unique
 914 sender and a unique receiver process. For multi-cast communication, i.e., non-irreflexive
 915 self-blocking c-actions, we need the even stronger notion of *constructive enabling* based on
 916 $\bar{iA}^*(R)$ instead of $\bar{iA}^\tau(R)$.

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

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

926 Obviously, to compute $iA^\tau(E)$ we need to know the full transition relation for E . In
 927 other words, the existence of a transition of a process P depends on the absence of certain
 928 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.

929 the behaviour of a part of P might depend on the behaviour of P itself. Does such a
 930 self-referential semantical definition make sense at all? How do we ensure the absence of a
 931 Russell-paradox style relation, by defining a process $\text{rec } p.P$ to have a transition to P' if it
 932 does not have a transition to P' for a suitable P' ? Such would be plainly inconsistent at
 933 the meta-level, because it would not only define simply an empty transition relation⁷, but it
 934 would not define a unique semantic relation at all.

935 In the standard theory of weakly prioritised process algebra as described in the cited
 936 references of CCS with priorities, the problem does not exist, because the set of (strong)
 937 initial actions $\text{iA}(E)$ can be determined by structural recursion on E without any reference
 938 to the transition relation. Unfortunately, for strong prioritized process algebra as defined in
 939 this paper, this is not possible anymore. So, how do we make sense of transition-generating
 940 rules (Par_i^*) that depend on negative premises? One option is to treat the negation in the
 941 side-condition $H \cap \overline{\text{iA}}^\tau(E) = \{\}$ constructively rather than classically, say like negation in
 942 normal logic programming. This would amount to building a second independent (inductive)
 943 rule system to generate constructive judgements about the absence of weakly reachable
 944 transitions. Trouble is that there is no canonical semantics of constructive negation, even in
 945 logic programming⁸.

946 Some progress has been made for unifying the proposals in a game-theoretic and
 947 intuitionistic setting which works well for special situations like the constructive semantics of
 948 Esterel [5] or StateCharts [17]. However, playing with constructive negation at the meta-level
 949 will give a variety of semantics whose adequacy might be difficult to judge in practical
 950 application contexts.

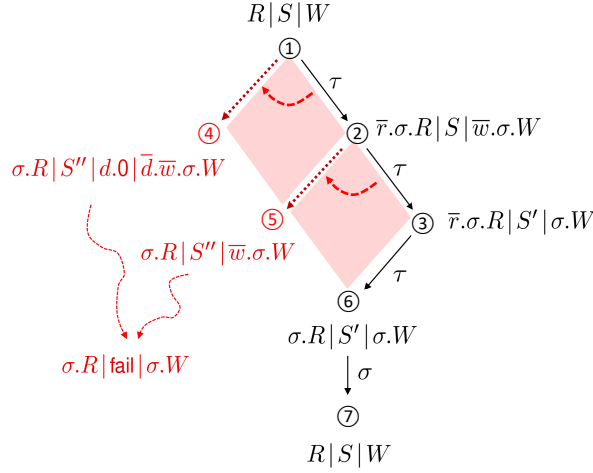
951 A more perspicuous attack is to keep the classical set-theoretic interpretation but use an
 952 over-approximation $L \cap \overline{\text{fiA}}^*(E) = \{\}$ where $\text{fiA}^*(E) \supseteq \text{iA}^\tau(E)$ is intuitively a superset of
 953 weakly reachable initial sender actions obtained from a relaxed transition relation, formally
 954 defined later in this paper. A natural over-approximation that can be defined independently
 955 is the “free speculative” scheduling that ignores all priorities. The stronger side-condition
 956 $L \cap \overline{\text{fiA}}^*(E) = \{\}$ forces the absence of a blocking sender action under the worst-case
 957 assumption of priority-free scheduling. Obviously, this is sound but will have the effect that
 958 some programs block that could make progress under a tighter approximation of the sets
 959 $\text{iA}^\tau(E)$. Between $\text{fiA}^*(E)$ and $\text{iA}^\tau(E)$ there be a whole range of different semantics that may
 960 or may not be equivalent to specific constructive semantics of negation. Exploring the range
 961 of options systematically will certainly be a worthwhile research exercise likely to cover much
 962 new ground in the theory of process algebras with priorities.

963 A.3 Synchronous CCS with (Strong) Priorities and Clocks is Confluent

964 We argue that the use of priorities for determinacy calls for a combination of strong priorities
 965 with a second important concept that has been investigated in the literature before, but
 966 independently. This is the well-known notion of a *clock*, from Timed Process Algebras, like
 967 e.g. TPL [18] and PMC [15]. Intuitively, a clock σ is a “broadcast action” in the spirit of
 968 Hoare’s CSP [19] that acts as a synchronisation barrier for a set of concurrent processes. It
 969 bundles each processes’ actions into a sequence of *macro-steps* that are aligned with each
 970 other in a lock-step fashion. During each macro-step, a process executes only a temporal slice
 971 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 $\text{rec } p.p$ 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.

972 *scope* to reach the end of the current phase. When all processes have reached the barrier,
973 they are released into the next round of computation.

974 This scheduling principle is well-known from computer hardware like sequential circuits [39]
975 and VHDL [20], or synchronous languages like Esterel [5], Lustre [16], SCCharts [41], BSP [40],
976 just to mention a few. For our purposes, clocks are the universal trick to break causality
977 cycles in priority scheduling, when processes need to communicate with each other in iterated
978 cycles (so-called *ticks* or *macro-steps*) while maintaining determinacy.

979 To get this off the ground we introduce a clock broadcast action σ to separate each round
980 of communication. We define the recursive processes $W \stackrel{\text{def}}{=} d.0|\bar{d}.w.\sigma.W$ and $R \stackrel{\text{def}}{=} \bar{r}.\sigma.R$
981 for one write and one read per macro-step. Assuming the content of the memory variable
982 is reset for every tick, we can define S as follows $S \stackrel{\text{def}}{=} w.S' + r:\{w\}.S'' + \sigma:\{w, r\}.S$ with
983 $S' \stackrel{\text{def}}{=} r.S'' + \sigma:\{r\}.S'$ and $S'' \stackrel{\text{def}}{=} w.\text{fail} + r:\{w\}.S'' + \sigma:\{w, r\}.S$. The clock always takes *lowest*
984 priority among all other rendez-vous actions out of a state. The idea is that the clock should
985 fire only if the system has stabilised and no other admissible data actions are possible. This
986 is called *maximal progress* and it is a standard assumption in timed process algebras [18, 11].
987 The fairly standard rules for clock actions, inspired by timed process algebras are then:

$$988 \frac{}{\sigma.P \xrightarrow{\sigma} \{\} P} (Clk_1) \quad \frac{P \xrightarrow{\sigma}_{R_1} P' \quad P \xrightarrow{\sigma}_{R_2} P' \quad (L_1 \cup L_2) \cap iA^\tau(R_1 | R_2) = \{\}}{P | Q \xrightarrow{\sigma}_{R_1 | R_2} L_1 \cup L_2 P' | Q'} (Clk_2)$$

989 Those two rules enforces global synchronisation and make the clock deterministic. As such,
990 the evaluation of $R|S|W$ “prunes” all reductions that could lead to non confluent reductions.
991 Fig. 7 shows how higher priorities can introduce determinism in scheduling. The importance
992 of the clock σ is that its associated synchronisation rule (Clk_2) enforces, in a quite natural way,
993 the barrier explicitly in the SOS rather than making it an artefact of a specific synchronisation
994 process, as such obliging the programmer to an extra effort. The fact that the broadcasting
995 clock is hard-wired into the rules can now be exploited for the computation of the *weak*
996 *initial action sets*, defined formally $iA^\tau(R)$. More precisely, we only need to predict the
997 behaviour of R for the current macro-step, i.e., up to the next synchronization of σ by R .
998 For a transition $P \xrightarrow{a}_R P'$ we know that all actions taking place after σ in R are definitely
999 observed to happen in the next macro step and thus not in competition with the current
1000 action a under blocking set L . In this fashion, the clock acts as a *stopper sentinel* for

XX:28 Strong Priority and Determinacy in Timed CCS

1001 prediction and evaluation of the blocking set L relative to the reduction environment R . If
1002 every recursion is instantaneous, then the search space becomes finite.

1003 As a final remark, for the purposes of this paper, adding synchrony and determinism in
1004 process algebras, we are not (and indeed we do not need to be) interested in higher-order
1005 calculi, like e.g. the π -calculus [29, 30], where processes are *first-class* objects, i.e. sent
1006 and received as a values, as in $a!P.Q$ and $a?x.(x|Q)$. In the next sections, we present the
1007 syntax, the structural operational semantics à la Plotkin, we set up all the definitions and
1008 the metatheory necessary to formally prove the claimed informal properties.

1009 **B** Examples, new and reloaded

1010 This section contains a number of reloaded and new examples in view of the definitions and
1011 results presented in the previous section.

1012 **Note on Conservativity**

1013 The weakly-enabled transitions of unclocked, free processes coincide with the semantics of
1014 genuine Milner's CCS [27], as stated in the following easy exercise.

1015 ▶ **Exercise 29** (On Def. 2). *If P is unclocked and free, then a transition $P \xrightarrow[R]{\alpha} Q$ is weakly
1016 enabled iff $P \xrightarrow{\alpha} Q$ is admissible, i.e., iff it is derivable in Milner's original SOS for CCS.*

1017 In fact, one can show that the semantics of weak enabling generates the theory of CCS^{Ph} [33].
1018 To be more precise, let us call a process P *irreflexive* if no action prefix $\alpha:H.Q$ occurring in
1019 P has $\alpha \in H$, i.e., blocks itself. In the semantics of CCS^{Ph} , such *self-blocking* prefixes do not
1020 contribute any transitions. Thus, CCS^{Ph} captures a theory of irreflexive processes. We leave
1021 as a less easy exercise that all weakly enabled transitions between irreflexive processes are
1022 precisely the ones allowed in CCS^{Ph} .

1023 ▶ **Exercise 30** (On Def. 2). *If P is unclocked and irreflexive, then a transition $P \xrightarrow[R]{\alpha} Q$ is
1024 weakly enabled iff $P \xrightarrow{\alpha} P'$ is derivable in the operational semantics of CCS^{Ph} .*

1025 Thus, from the point of view of CCS^{spt} , theories of CCS and CCS^{Ph} correspond to theories
1026 of free and irreflexive processes, respectively, under weak enabling. Here we suggest two
1027 extensions to this classical work. Firstly, we apply a stronger scheduling restrictions, called
1028 *strong enabling* (Def. 28 below) and *constructive enabling* (Def. 3 above) to achieve confluence
1029 and secondly we make use of self-loops to control sharing.

1030 **Priorities and Scheduling**

1031 ▶ **Example 31** (On Def. 12). If $A \not\equiv B$ then $a.A + a.B$ is not coherent while $a:a.A|a:a.B$
1032 is coherent. Conflicting choices on the same label in a single thread cannot be externally
1033 observed and thus cannot be deterministically scheduled via their externally observable
1034 properties. In coherent processes such choices are assumed to be resolved internally by each
1035 thread itself, rather than through the precedence relation. Note that the process $a:a.A + a:a.B$
1036 in which the two action prefixes are self-blocking, is not coherent either. The reason is that
1037 the two identical (self-blocking) transitions lead to structurally distinct states $A \not\equiv B$. Thus,
1038 the confluence condition expressed in Def. 12 applies and A and B should be reconvergent
1039 with transitions $A \xrightarrow{a} Q'$ and $B \xrightarrow{a} Q'$. But this is not guaranteed for arbitrary A and B . On

1040 the other hand, a single self-blocking prefix $a:a.A$ is coherent. Without the self-blocking, $a.A$
 1041 is only coherent if process A permits infinite repetition of a as in $A \stackrel{def}{=} a.A$ or $A \stackrel{def}{=} a | A$.

1042 The precedence relation cannot resolve the choice of a prefix with itself. The precedence
 1043 relation can resolve the choice between *distinct* actions prefixes. For instance, if $a \neq b$ then
 1044 $a.A + b.B$ is not coherent, for any A and B . However, both $a:a.A | b:b.B$ and $a:a.A + b:\{a, b\}.B$
 1045 are coherent. In the former case, the actions are concurrently independent and so confluence
 1046 occurs naturally. In the latter case, the choice is resolved by precedence: No reconvergence
 1047 is required by coherence Def. 12 as the actions are interfering with each other. ◀

1048 ▶ **Example 32 (On Normal form).** For the process $P_1 \stackrel{def}{=} (a.c | b.d | \bar{a} | \bar{b}) \setminus \{a, b\}$ we have
 1049 $(a.c | d | \bar{a}) \setminus \{a, b\} \stackrel{\tau}{\leftarrow} P_1 \stackrel{\tau}{\rightarrow} (c | b.d | \bar{b}) \setminus \{a, b\}$ where the continuation processes $(c | b.d | \bar{b}) \setminus \{a, b\}$
 1050 and $(a.c | d | \bar{a}) \setminus \{a, b\}$ are not structurally equivalent. They are not even behaviourally
 1051 equivalent. However, both reductions are concurrently independent and commute with each
 1052 other. We can execute both rendezvous synchronisations $\tau = a | \bar{a}$ and $\tau = b | \bar{b}$ in any order,
 1053 the process P converges to a unique final outcome $(c | d) \setminus \{a, b\}$. Similarly, in the multi-cast
 1054 process $P_2 \stackrel{def}{=} (a.c | a.d | \bar{a} | \bar{a}) \setminus \{a\}$ the reduction behaviour is not structurally deterministic
 1055 but nevertheless generates a unique normal form. ◀

▶ **Example 33 (On Sequentiality à la Esterel in CCS^{spt}).** Like CCS, our calculus CCS^{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 \bar{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

$$[P_1 | P_2]_t \stackrel{def}{=} ([P_1]_{s_1} | [P_2]_{s_2} | s_1:s_1.s_2:s_2.\bar{t}) \setminus \{s_1, s_2\}$$

where s_1 and s_2 are fresh “local” termination signals. The “termination detector” $T_{seq} \stackrel{def}{=} s_1:s_1.s_2:s_2.\bar{t}$ first waits for P_1 to terminate and then for P_2 , before it sends the signal \bar{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*

$$T_{csum} \stackrel{def}{=} s_1:s_1.s_2:s_2.\bar{t} + s_2:s_2.s_1:s_1.\bar{t}$$

1056 as discussed in Milner [27] (Chap. 11.4). This does not work, because the sum must
 1057 explicitly unfold all possible orders in which the termination signals can arrive and thus
 1058 again creates an exponential descriptive complexity. Interestingly, in CCS^{spt} we can exploit
 1059 precedence scheduling to obtain a robust and compositional solution. Consider the process
 1060 $T_{par} \stackrel{def}{=} s.T_{par} + \bar{t}:\{s, \bar{t}\}$. It engages in an unbounded number of receptions on channel s
 1061 without changing its state. At the same time, it offers to send the termination signal \bar{t} ,
 1062 but only if the action s is not possible. Under the weak enabling strategy, the composition
 1063 $\bar{s} | \bar{s} | T$ will consume the two senders \bar{s} and only then engage in t . In this way, we can use
 1064 T_{par} to detect the *absence* of s which corresponds to the termination of the two environment
 1065 processes $\bar{s} | \bar{s}$, in any order. The signal s could be the shared termination signal of each

XX:30 Strong Priority and Determinacy in Timed CCS

1066 process P_1 and P_2 that is sent and consumed by T_{par} if and when the process terminates. If
 1067 we execute the parallel composition $P_1 | P_2 | T_{par}$ under constructive enabling, the termination
 1068 signal \bar{t} will not be sent until such time that both P_1 and P_2 have been able to send s to
 1069 T . From the scheduling point of view, the composition $P_1 | P_2$ acts like a counter that is
 1070 initialised to 2, representing the number of potential offerings of s initially in $P_1 | P_2$. It
 1071 gets decremented each time one of P_1 or P_2 terminates. For instance, $0 | P_2$ and $P_1 | 0$ have
 1072 one potential offering left, while $0 | 0$ has no more potential to send s . At this moment, the
 1073 counter has reached 0 and $0 | 0 | T_{par}$ can take the \bar{t} action and thereby send termination.
 1074 In sum, there are two important observations here: First, we find that precedences with
 1075 constructive enabling permits us to code sequential composition of Esterel in a compositional
 1076 fashion. Second, the coding of Milner's confluent sum T_{csum} is possible in CCS^{spt} with linear
 1077 descriptive complexity. \blacktriangleleft

1078 The next example illustrates that the closure property for coherence of hiding (Thm. 14(6),
 1079 see also Prop. 65) needs to require that the hidden clock must not take precedence over
 1080 rendezvous actions.

► **Example 34** (On Clock-hiding). The process $P \stackrel{def}{=} \sigma.A + a:\sigma.B | \sigma$ offers a choice of two
 transitions $P \xrightarrow{\sigma}_{\{\}} A$ and $P \xrightarrow{a}_{\{\sigma\}} B | \sigma$. These transitions interfere with each other so that the
 σ -transition blocks the a -transition under weak enabling, given that $\{\sigma\} \cap \bar{iA}(\sigma) = \{\sigma\} \neq \{\}$.
 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/σ 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

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

1081 where the blocking set of the a -transition turns into $\{\sigma\} - \{\sigma\} = \{\}$. Now, because σ is
 1082 removed from the blocking set, the a -transition is even constructively enabled. Moreover,
 1083 both the τ -transition and the a -transition are interference-free. However, in general, there is
 1084 no chance that we can find reconvergent transitions $A/\sigma \xrightarrow{a} Q_1$ and $(B|\sigma)/\sigma \xrightarrow{\tau} Q_2$ with
 1085 $Q_1 \equiv Q_2$. Thus, P/σ is not coherent any more. \blacktriangleleft

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:

$$\frac{P \xrightarrow{\alpha}_{R} Q \quad H' = H - \{\sigma\} \cup \{\tau \mid \sigma \in \bar{iA}^*(R) \cap H\}}{P/\sigma \xrightarrow{\alpha}_{R/\sigma} Q/\sigma} \text{ (Hide}^*)$$

In this manner, the enabling check for the conclusion transition verifies that $\tau \notin H'$ and
 thus that $\sigma \notin \bar{iA}^*(R) \cap H$. 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

$$\frac{P \xrightarrow{\alpha}_{R} Q \quad \alpha \notin \{\ell, \bar{\ell}\} \quad H' = H - \{\ell, \bar{\ell}\} \cup \{\tau \mid \ell \in \bar{iA}^*(R) \cap H\}}{P \setminus \ell \xrightarrow{\alpha}_{R \setminus \ell} Q \setminus \ell} \text{ (Restr}^*)$$

1086 In this paper we stick to the simpler versions (*Hide*) and (*Restr*) but conjecture that the side
 1087 conditions on the policy in Thm. 24(6,7) can be lifted to show that for (*Hide**) and (*Restr**)
 1088 the operations of hiding and restriction preserve coherence.

► **Example 35** (Clock-hiding with built-in Scheduling Test). Consider the process $P \stackrel{def}{=} \sigma.A + a:\sigma.B \mid b.\sigma$ 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 \bar{b} . As in the previous example P/σ would not be coherent. The extended hiding rule (*Hide**) would generate the a -transition

$$P/\sigma \xrightarrow{(b.\sigma)/\sigma\{\tau\}} (B \mid b.\sigma)/\sigma \text{ from } P \xrightarrow{a/b.\sigma\{\sigma\}} A \mid b.\sigma.$$

1089 The τ enters into the blocking set because of the race test which discovers that $\sigma \in \bar{iA}^*(b.\sigma) \cap$
 1090 $\{\sigma\}$, where $\{\sigma\}$ is the blocking set of the a -transition in P . Hence, the a -transition of P/σ
 1091 is not (constructively) enabled anymore, which saves coherence. There is no requirement to
 1092 exhibit any reconvergence for it. ◀

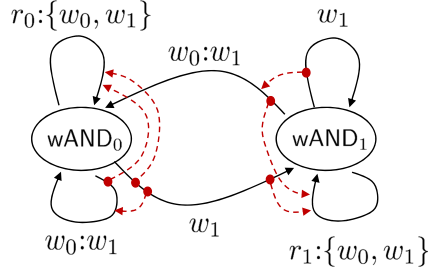
1093 Policies

1094 ► **Example 36** (On coherence and pivotable processes). The process $P_1 \stackrel{def}{=} a \mid \bar{b} \mid \bar{a}:b + b$ is
 1095 coherent and pivotable with a policy π such that $\pi \Vdash b \dashrightarrow \bar{a}$ and $\pi \Vdash \bar{b} \diamond a$. The rendezvous
 1096 synchronisation $\tau = a \mid \bar{a}$ is blocked by the presence of the offering \bar{b} from a *third* thread. We
 1097 have $P_1 \xrightarrow{\tau/b\{\bar{b}\}} \bar{b}$, which is not weakly enabled, because $\{b\} \cap \bar{iA}(\bar{b}) \neq \{\}$. The blocking is due
 1098 to the concurrent context \bar{b} of the transition. For contrast, the process $P_2 \stackrel{def}{=} (a + \bar{b}) \mid (\bar{a}:b + b)$
 1099 is pivotable but not coherent. In our semantics, the synchronisation of a and $\bar{a}:b$ in P_2 is
 1100 blocked by the race test of rule (*Com*) which introduces τ into the blocking set $P_2 \xrightarrow{\tau/0\{b,\tau\}} 0$
 1101 because $\{b\} \cap \bar{iA}(a + \bar{b}) \not\subseteq \{\bar{a}\}$. This transition therefore is not weakly enabled, because
 1102 of $\{b,\tau\} \cap (\bar{iA}(0) \cup \{\tau\}) \neq \{\}$. We can make the pivotable P_1 also coherent by forcing a
 1103 sequential ordering between a and \bar{b} in the first thread, say $P_3 \stackrel{def}{=} a.\bar{b} \mid (\bar{a}:b + b)$, then the
 1104 synchronisation is even strongly enabled $P_3 \xrightarrow{\tau/0\{b\}} \bar{b}$. If we introduce a precedence into the
 1105 first thread, say $P_4 \stackrel{def}{=} (a:b + \bar{b}) \mid (\bar{a}:b + b)$, then we are also coherent but no longer pivotable.
 1106 Like for P_2 , the synchronisation $\tau = a \mid \bar{a}$ is blocked and not weakly enabled, because of the
 1107 introduction of τ into the blocking set by the race condition of (*Com*). ◀

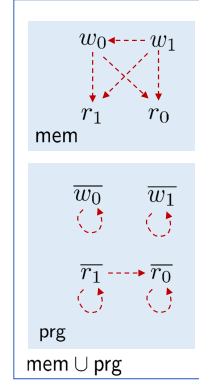
1108 ► **Example 37** (On Coherent and Pivotable Processes). The process $P_1 \stackrel{def}{=} \sigma + a:\sigma \mid \bar{a}$ is
 1109 coherent and pivotable, with policy $\sigma \dashrightarrow a \in \pi$ and $\sigma \diamond \bar{a} \in \pi$. It has both $a, \bar{a} \in iA(P)$
 1110 but $\sigma \notin iA(P)$ since the clock is not offered by the parallel process \bar{a} . The synchronisation
 1111 $\tau = a \mid \bar{a}$ can go ahead. On the other hand, the process $P_2 \stackrel{def}{=} \sigma + a:\sigma \mid \bar{a} + \sigma:\bar{a}$ has all three
 1112 actions $\sigma, a, \bar{a} \in iA(P_2)$ initial. It is coherent but not pivotable. Since both a and \bar{a} are in
 1113 a precedence relationship with σ , P_2 does not satisfy concurrent independence of σ with
 1114 neither a nor \bar{a} . Formally, any policy π for P_2 must have $\sigma \dashrightarrow a \in \pi$ and $\bar{a} \dashrightarrow \sigma \in \pi$ violating
 1115 the pivot requirement that if $\sigma \dashrightarrow a \in \pi$, then $\sigma \diamond \bar{a} \in \pi$. The same is true of the process
 1116 $P_3 \stackrel{def}{=} \sigma + (a:\sigma \mid \bar{a})$ which can perform both σ and a (silent) reduction $\tau = \alpha \mid \bar{a}$ that has σ in
 1117 its blocking set. ◀

1118 An interesting special class of pivot policies are input-scheduled policies.

1119 ► **Definition 38** (Input-scheduled Processes). A policy π is called *input-scheduled* if $\pi \subseteq \pi_{is}$
 1120 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
 1121 $\ell_1 = \ell_2$ or $\ell_1, \ell_2 \in \mathcal{A} \cup \mathcal{C}$. A process is *input-scheduled* if P conforms to an input-scheduled
 1122 policy.



(a) A Boolean Read-Write Memory with coherent states $wAND_0 : mem$ and $wAND_1 : mem$.



(b) The $mem \cup prg$ Pivot Policy.

■ **Figure 8** A boolean memory process.

1123 ▶ **Example 39** (On Pivotal but not Input-scheduled Processes). The process $a + b:c \mid \bar{a} + \bar{b}:\bar{a}$
 1124 is pivotal. It conforms to π with precedences $c \dashrightarrow b \in \pi$ and $\bar{a} \dashrightarrow \bar{b} \in \pi$, while at the same
 1125 time $\bar{c} \diamond \bar{b} \in \pi$ and $a \diamond b \in \pi$. However, assuming $a \neq b$, the process is not input-scheduled,
 1126 because it has $\bar{a} \dashrightarrow \bar{b} \in \pi$ while $\bar{a} \diamond \bar{b} \in \pi_{is}$. The same is true of the process in Ex. 5. ◀

1127 ▶ **Example 40** (On Read/Write Process, reloaded). Fig. 8 show that the processes $wAND_v$
 1128 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,
 1129 mem is a pivot policy. The processes $wAND_v$ are also coherent. One verifies by simple case
 1130 analysis that for any derivative Q of $wAND_v$, if $Q \xrightarrow{\ell}_H Q'$ and $\ell \notin H$ then there is a repetition
 1131 $Q' \xrightarrow{\ell}_H Q''$ with $H' = H$. This yields Def. 12. Secondly, suppose $Q \xrightarrow{\ell_1}_{H_1} Q_1$ and $Q \xrightarrow{\ell_2}_{H_2} Q_2$.
 1132 Then, obviously, if $\ell_1 \neq \ell_2$, the two transitions are never interference-free, because of the
 1133 choice of blocking sets which orders all summation prefixes in a linear precedence order. If
 1134 $Q_1 \not\equiv Q_2$ then necessarily $\ell_1 \neq \ell_2$. Thus, Def. 12 is satisfied, too, for trivial reasons. So, the
 1135 memory, which satisfies $wAND_v \Vdash mem$, is pivotal.

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 \bar{w}_i for writing and \bar{r}_i for reading. Each writing action of P is a self-blocking prefix $\bar{w}_i:\bar{w}_i.Q$ and each reading is a choice $\bar{r}_1:\bar{r}_1.R_1 + \bar{r}_0:\{\bar{r}_0, \bar{r}_1\}.R_0$ 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 $\bar{r}_1 \dashrightarrow \bar{r}_i \in prg$ and $\bar{r}_0 \dashrightarrow \bar{r}_0 \in prg$. Now we observe that the union $mem \cup prg$ of policies is a pivot policy. Since $wAND_v \Vdash mem$ and $P \Vdash prg$ we have $wAND_v \mid P \Vdash mem \cup prg$ as well as $P \Vdash mem \cup prg$. Then, by Prop. 64 we get

$$wAND_v \mid P \Vdash mem \cup prg$$

1136 i.e., the composition is coherent and pivotal. In fact, we can put the memory in
 1137 parallel with arbitrarily many sequential programs coherent for prg and the composition
 1138 $wAND_v \mid P_1 \mid P_2 \mid \dots \mid P_n$ will be coherent for $mem \cup prg$. In particular, all reductions will be

1139 confluent under strong enabling. Technically, what happens is that first all writers of value
 1140 1, synchronising as $w_1 \mid \bar{w}_1$, will go ahead. Then all writers of 0, synchronising as $w_0 \mid \bar{w}_0$
 1141 become enabled. Only then the readers will be able to retrieve the final value via $r_0 \mid \bar{r}_0$ or
 1142 $r_1 \mid \bar{r}_1$ rendezvous synchronisations. ◀

1143 Observe that the pivot policy $\text{mem} \cup \text{prg}$ of Ex. 40 permits a choice (and precedence)
 1144 between both the receiver actions w_0, w_1 but also between the sender actions \bar{r}_0, \bar{r}_1 . As
 1145 a result, the combined memory and program, which is coherent for $\text{mem} \cup \text{prg}$, is not an
 1146 input-scheduled process in the sense of Def. 38. This breaks the assumption on which the
 1147 theory of classical prioritised process algebras like [10] or [36] are based. In reference to the
 1148 former, Phillips writes:

1149 *They [Camilletti and Winskel] decide to sidestep this question by breaking the symmetry*
 1150 *in CCS between inputs and outputs, and only allowing prioritised choice on input*
 1151 *actions. We feel that this complicates the syntax and operational semantics, and should*
 1152 *not be necessary. [33]*

1153 Our example and analysis indeed highlights the usefulness of the more general setting
 1154 suggested by Phillips. In order to explain the theory of deterministic shared memory we
 1155 need a larger class of processes than those considered in CCS^{CW} .

1156 Note that all the examples of Sec. 3 have the “input-scheduled” property that (non-trivial,
 1157 i.e. non-self-blocking) precedences only exist between receiver actions \mathcal{A} , while sender actions
 1158 $\bar{\mathcal{A}}$ never appear in any choice contexts. These are the processes covered by [10] which restrict
 1159 prioritised choice to receiver actions (“input-guarded, prioritised choice”) in the same way as
 1160 the programming language Occam [4] with its “PRIALT” construct does (cf. [10]).

1161 ▶ **Example 41.** The memory cell schedules readers after the writers and among the writers
 1162 prefers to engage first with those that set the value to 0, in the prioritised sum. This
 1163 implements an Esterel-style resolution function for multi-writer convergecast, multi-reader
 1164 broadcast of boolean signals. All concurrent readers receive the same value, which is the
 1165 combined-AND of all values written by any concurrent writer. If any process writes a boolean
 1166 1 it will win the competition. This is also known as a “wired-AND”. Likewise, we can
 1167 implement a memory cell that models a “wired-OR” if we swap the roles of w_0 and w_1 in
 1168 the prioritised choices of wAND_v .

1169 Note that the wired-AND behaviour only applies to concurrent threads. A single thread
 1170 can use the wAND cell as a local memory and destructively update it sequentially as it
 1171 wishes. Consider the difference: Both *concurrent* writers $\bar{w}_1:\bar{w}_1 \mid \bar{w}_0:\bar{w}_0 \mid \text{wAND}_0 \mid R$ and
 1172 $\bar{w}_0:\bar{w}_0 \mid \bar{w}_1:\bar{w}_1 \mid \text{wAND}_0 \mid R$ will necessarily make the reader $R \stackrel{\text{def}}{=} r_0:r_1.A + r_1:\{r_0, r_1\}.B$ detect
 1173 a value r_0 and therefore end in state A (wired-AND of w_0 and w_1). The two single-threaded
 1174 *sequential* writings $\bar{w}_1:\bar{w}_1.\bar{w}_0:\bar{w}_0 \mid \text{wAND}_0 \mid R$ and $\bar{w}_0:\bar{w}_0.\bar{w}_1:\bar{w}_1 \mid \text{wAND}_0 \mid R$ will generate two
 1175 different readings by R : The first is the reading r_0 , leading to A . The second is the reading
 1176 r_1 , leading to B . ◀

1177 Strong Enabling vs Constructive Enabling

1178 As an intermediate step to implement our precedence-based scheduling, the previous example
 1179 suggests we look at the *weak initial actions* for a process. The next example shows why
 1180 strong enabling, based on $\bar{iA}^\tau(R)$, is not sufficient and why we need the stronger notion of
 1181 c-enabling (Def. 3), based on $\bar{iA}^*(R)$.

XX:34 Strong Priority and Determinacy in Timed CCS

1182 ▶ **Example 42** (On Multi-Cast and Constructive Enabling). Suppose $S_0 \stackrel{\text{def}}{=} a:e.S_0 + e.S_1 +$
 1183 $\sigma:\{a, e\}.S_0$ is an Esterel Signal (Sec. 9) in its absent state, with abbreviations $e = \text{emit}$,
 1184 $a = \text{abs}$ and $p = \text{pres}$. Its initial action $S_0 \xrightarrow{a}_{0|\emptyset} S_0$ to communicate absence does not block
 1185 itself to permit multi-reader scenarios. A typical reader is of form $R \stackrel{\text{def}}{=} \bar{p}:\bar{p}.R_1 + \bar{a}:\{\bar{a}, \bar{p}\}.R_0$.
 1186 This is a choice: if the signal is present we execute R_1 , otherwise if it is absent we execute
 1187 R_0 . The two sender actions \bar{p} and \bar{a} are self-blocking, because these prefixes can only be
 1188 consumed once. In essence, what we want is that there can be many senders of the value
 1189 labels \bar{p}, \bar{a} (readers) but at most one receiver for p, a (signal).

1190 Consider a reader $C \stackrel{\text{def}}{=} \bar{a}:\bar{a}.\bar{e}:\bar{e}.C_1$ that wants to emit the signal when it finds it absent.
 1191 The composition $S_0 | C$ is deterministic under strong enabling. $S_0 | C \xrightarrow{\tau}_{0|\emptyset} S_0 | \bar{e}:\bar{e}.C_1$.
 1192 The blocking set contains two labels:

- 1193 (1) First, $e \in \{e, \bar{a}\}$ indicates priority, i.e., that we do not want another concurrent thread
 1194 sending \bar{e} , because the synchronisation for absence is based on the assumption that
 1195 there is no emission. Specifically, $S_0 | C | \bar{e} \xrightarrow{\tau}_{0|\emptyset} S_0 | \bar{e}:\bar{e}.C_1 | \bar{e}$ will block because
 1196 $\{e, \bar{a}\} \cap \bar{iA}^\tau(0|0|\bar{e}) \neq \{\}$. Instead, the transition $S_0 | C | \bar{e} \xrightarrow{\tau}_{0|C|0} S_1 | C | 0$ which emits
 1197 the signal can proceed.
- 1198 (2) Second, $\bar{a} \in \{e, \bar{a}\}$ indicates that the sending prefix $\bar{a}:\bar{a}$ of the reader C (which has entered
 1199 into the synchronisation $\tau = a|\bar{a}$) must not be consumed by another competing receiver.
 1200 Specifically, $S_0 | C | a \xrightarrow{\tau}_{0|\emptyset} S_0 | \bar{e}:\bar{e}.C_1 | a$ will block.

1201 Now, take $S_0 | C | D$ where the signal S_0 is connected with two readers, where C is as above
 1202 and $D \stackrel{\text{def}}{=} \bar{a}:\bar{a}.\bar{e}:\bar{e}.D_1$ analogous. Both readers can go ahead and synchronise in reactions
 1203 $(\tau = a|\bar{a}) S_0 | C | D \xrightarrow{\tau}_{0|\emptyset} S_0 | \bar{e}:\bar{e}.C_1 | D$ and $S_0 | C | D \xrightarrow{\tau}_{0|C|0} S_0 | C | \bar{e}:\bar{e}.D_1$ which
 1204 are both strongly enabled since $\bar{iA}^\tau(0|0|D) = \{a\} = \bar{iA}^\tau(0|C|0)$ which is disjoint from the
 1205 blocking set $\{e, \bar{a}\}$. This may seem ok, as we wanted to have multiple readers access the signal
 1206 on the same port a . However, now the emission of \bar{e} by the process that advanced first will
 1207 block the reading of absence by the other which stayed behind. In each case symmetrically,
 1208 there is only one (strongly enabled) transition left, viz. $S_0 | \bar{e}:\bar{e}.C_1 | D \xrightarrow{\tau}_{0|\emptyset} S_1 | C_1 | D$
 1209 and $S_0 | C | \bar{e}:\bar{e}.D_1 \xrightarrow{\tau}_{0|C|0} S_1 | C | D_1$ where the final states $C_1 | D$ and $C | D_1$ of the readers
 1210 are behaviourally different. Thus, we have non-determinism while the confluence is lost.

1211 The core of the problem is that the concurrent contexts $0|0|D$ and $0|C|0$, respectively, of
 1212 the above rendezvous synchronisations and do not account for the fact that the signal S_0 can
 1213 be simultaneously reused by the other reader that is not involved in the transition. We could
 1214 account for this reuse by adding S_0 into the concurrent context as in $S_0 | C | D \xrightarrow{\tau}_{S_0|0|D} S_0 | \bar{e}:\bar{e}.C_1 | D$.
 1215 Now D can interact with S_0 in the concurrent context $S_0 | 0 | D$. The above
 1216 transition depends on the assumption that no other thread will send \bar{e} . This is not true,
 1217 because now $e \in \bar{iA}^\tau(S_0 | 0 | D)$. Thus, putting the shared signal S_0 into the concurrent
 1218 environment is enforcing determinism by blocking the strong enabling properties of the above
 1219 transitions. Unfortunately, the fix is too strong. The presence of S_0 in the concurrent context
 1220 would block even the single reader in $S_0 | \bar{a}:\bar{a}$. Specifically, the reduction $S_0 | \bar{a}:\bar{a} \xrightarrow{\tau}_{S_0|0} S_0 | 0$
 1221 $S_0 | 0$ would block, because $\bar{a} \in \bar{iA}(S_0) \subseteq \bar{iA}^\tau(S_0 | 0)$, and so the reduction is not strongly
 1222 enabled any more. Thus, our fix would not permit any reading of the signal S_0 , whatsoever.

1223 The solution is to tighten up the notion of strong enabling and close it under arbitrary non-
 1224 clock transitions in the concurrent environment rather than just the silent steps. Technically,
 1225 we define a larger set $\bar{iA}^*(E) \supseteq \bar{iA}^\tau(E)$ of potential labels (Def. 3) that closes under *all* non-
 1226 clock actions of E . In this case, we do find $e \in \{e, \bar{a}\} \cap \bar{iA}^*(0|0|D) \neq \{\}$. In this way, diverging
 1227 reductions are no longer enabled. On the other hand, the reduction $S_0 | C \xrightarrow{\tau}_{0|\emptyset} S_0 | \bar{e}:\bar{e}.C_1$

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

■ **Figure 9** Simplified version of ABRO.

1228 remains enabled, because $\bar{iA}^*(0 \mid 0) = \{\}$. In general, single readers $S_0 \mid C$ where C is an
1229 arbitrary sequential process, or multiple readers $S_0 \mid C \mid D$ where *at most one* of C or D is
1230 writing, i.e., sending \bar{e} , remain enabled. ◀

1231 ▶ **Example 43** (On Fully Multi-Cast ABRO). To model full multi-cast ABRO in CCS^{spt} , we
1232 could use sharable (i.e., non-self-blocking) prefixes on internal actions to show how they
1233 are useful to model the control-flow of Esterel programs, specifically termination and reset.
1234 Consider the process ABRO from Ex. 10 again:

1235 ■ The reset watchdog R can receive a reset signal r or silently move to state $\sigma.R$ where it
1236 synchronises on the clock σ to repeat in the next macro-step. When a reset r is received by
1237 the environment, then R executes the sequence $r:r.\bar{k}_1:\bar{k}_1.\bar{k}_2:\bar{k}_2.\bar{k}_3:\bar{k}_3.\bar{k}_4:\bar{k}_4.ABRO$ whereby
1238 it sends a “kill” signal \bar{k}_i to each of the four threads A , B , T and O , in some arbitrary
1239 but fixed order. When all kills have been delivered, the watchdog R will restart ABRO.
1240 Until such time, there is no clock possible. The second transition $R \xrightarrow{\tau}_{\{r\}} \sigma.R$ preempts
1241 this reset choice but has lower priority as it is blocked by r . It is only taken when there
1242 is no reset possible, i.e., when the reset signal r is absent. Note that with the preemption
1243 of the the kill signals \bar{k}_i are removed from the potential, because $\bar{iA}^*(\sigma.R) = \{\}$, i.e., we
1244 do not look past the clock prefix.

1245 ■ The A and B processes are analogous to each other with the role of a and b swapped.
1246 Looking at A , we see that it has a choice of three actions, viz to receive a kill signal k_1 ,
1247 an input signal a or perform the clock action σ to start the next macro step. The kill k_1
1248 has highest priority, action a second and the clock can only go ahead if there is neither a
1249 k_1 nor an a signal to be received. When there is no kill but an a signal, then the process
1250 A sends an \bar{s} signal to indicate its termination. When the kill k_1 occurs, it enters into the
1251 halt state 0 where it is inactive but permits any number of clock ticks. Since this is the
1252 neutral element for parallel composition, A essentially drops out of the picture. Note that
1253 we cannot use the inactive 0 here, because then the terminated process A would block
1254 the clock for all other threads. Finally, when there is neither a kill k_1 nor signal a , the A
1255 process can synchronise on σ and repeat as A in the next macro-step. In this fashion, A
1256 will patiently wait for a clock tick in which k_1 or a occur. If both occur, the kill takes
1257 priority: For instance, the transition $A \mid \bar{k}_1.0 \mid \bar{a}.0 \xrightarrow{\tau}_{0 \mid 0 \mid \bar{a}.1}^{\{k_1\}} 0 \mid 0 \mid \bar{a}.0$ is constructively
1258 enabled while $A \mid \bar{k}_1.0 \mid \bar{a}.0 \xrightarrow{\tau}_{0 \mid \bar{k}_1.0 \mid 0}^{\{k_1, a\}} 0 \mid \bar{k}_1.0 \mid 0$ is not enabled.

1259 ■ The process T is the termination detector implementing the principle discussed above.
1260 It forms a clock-patient, killable counter that waits for any possible reception of signal
1261 s before it sends termination \bar{t} to process O . Note that T itself does not need to know
1262 how many possible termination signals there are. It waits for the termination count

XX:36 Strong Priority and Determinacy in Timed CCS

1263 (implemented by iA^*) to reach zero.
 1264 ■ When process O receives the termination signal t it sends the output signal \bar{o} to the
 1265 environment and then halts. Alternatively, with highest priority, it can be killed by
 1266 reception of k_5 or engage in the clock σ if neither a kill k_5 nor a termination t is present.
 1267 The reduction $\tau = \bar{t}:\{k_3, \bar{t}, s\} | t:\{k_4, t\}$ between T and O will be blocked by precedence,
 1268 for as long as at least one of the processes A or B has a choice term that sends \bar{s} , i.e., as
 1269 long as $s \in i\bar{A}^*(A|B)$. Only when both prefix sequences $a:\{a, k_1\}.\bar{s}:\bar{s}.0$ and $b:\{b, k_2\}.\bar{s}:\bar{s}.0$
 1270 have been eliminated by reductions, then the termination can occur.

1271 Using multi-cast, we can improve on this as seen in Fig. 9: We can integrate the termination
 1272 test into the output process O if we let O loop on consumption of s rather than T . Similarly,
 1273 we can use only a single (multi-cast) kill signal k shared between A, B, O : The reset process
 1274 R keeps sending \bar{k} unboundedly often, until all target processes have received a copy and all
 1275 receptions k have been consumed. At that point, it takes a clock step to repeat ABRO from
 1276 the start.

1277 One can verify that ABRO is coherent and pivotable using Thm. 24.

1278 ■ The choices in each of the threads A, B, R, O are fully resolved by the blocking sets,
 1279 i.e., every pair of actions in the summations are distinct and in precedence relationships.
 1280 Every rendezvous prefix is self-blocking unless the action can be repeated with the same
 1281 blocking in the continuation process. This happens for channels s and k in the recursions
 1282 $O \stackrel{def}{=} \dots + s:k.O + \dots$ and $R' \stackrel{def}{=} \bar{k}.R' + \dots$. Thus, the threads A, B, R, O themselves are
 1283 coherent.

1284 ■ A suitable policy π , for which the threads A, B, R, O are coherent, enforces the precedences
 1285 ■ $k \dashrightarrow \ell \in \pi$ for $\ell \in \{k, a, b, s, t, \sigma\}$
 1286 ■ $\bar{\ell} \dashrightarrow \bar{\ell} \in \pi$ for $\bar{\ell} \in \{\bar{r}, k, a, b, \bar{s}, t, \bar{o}\}$
 1287 ■ $s \dashrightarrow t \in \pi$
 1288 ■ $\bar{\ell} \dashrightarrow \sigma \in \pi$ for $\bar{\ell} \in \{a, b, s, t\}$.

1289 The dual matching pairs for these precedences, however, are all concurrently independent.
 1290 More precisely, we look at distinct labels, $\bar{k} \diamond \bar{\ell} \in \pi$ for all $\bar{\ell} \in \{\bar{r}, o, a, b, s, t, \sigma\}$, $\bar{s} \diamond \bar{t} \in \pi$
 1291 and $\bar{\ell} \diamond \sigma \in \pi$ for $\bar{\ell} \in \{\bar{r}, o, a, b, s, t\}$.

1292 ■ Thus, the policy π (and thus $\pi \setminus \mathcal{R}$) is a pivot policy and so the parallel composition
 1293 $A|B|R|O$ is coherent for π . Further, π is precedence-closed for $\{s, t\}$ which means that
 1294 $(A|B|R|O) \setminus \{s, t\}$ is coherent for $\pi \setminus \{s, t\}$. Finally, since $\{\} \subseteq \{\bar{\ell} \mid \bar{\ell} \dashrightarrow \sigma \in \pi \setminus \{s, t\}\} =$
 1295 $\{a, b\}$, the process ABRO (which starts with a prefix $\sigma:H$ where $H = \{\}$) is coherent for
 1296 $\pi \setminus \{s, t\}$. ◀

1297 **C** Auxiliary Results

1298 In the following we list some simple observation on the properties of $iA(P)$, $iA^\tau(P)$ and
 1299 $iA^*(P)$ that will be used in the proofs.

1300 ▶ **Lemma 44.** *Let P, Q be processes:*

- 1301 (1) $P \xrightarrow[R]{\sigma} Q$ implies $R \equiv 0$.
- 1302 (2) $\alpha \in iA(P)$ iff $P \xrightarrow{\alpha} Q$.
- 1303 (3) $iA(P) \cap \mathcal{R} \subseteq iA(P|Q)$.
- 1304 (4) If P is single-threaded, then $P \xrightarrow[R]{\alpha} Q$ implies $\alpha \in \mathcal{L}$, $R \equiv 0$ and $\tau \notin H$.
- 1305 (5) If $P \xrightarrow[R]{\alpha} Q$ then $iA(R) \subseteq iA(P)$.

1306 **Proof.** (2) is by definition. The other points follow by an inspection of the SOS rules, using
1307 the definition of $iA(P)$. ◀

1308 Lem. 44(1) means that clock actions always run in an empty concurrent environment. Observe
1309 that Lem. 44(2) means that the set $iA(P)$ corresponds to the initial actions of P . The presence
1310 of τ in the blocking set of a transition provides information about the set of initial actions of
1311 a process. Specifically, in rule (*Com*) the presence of τ in the set $H = \text{race}(P|Q)$ indicates
1312 a blocking situation arising from the priorities in P or Q conflicting with the rendezvous
1313 actions ℓ or $\bar{\ell}$. Thus, the condition $\tau \notin H$ can be used to implement priority-based scheduling.
1314 Lem. 44(4) implies that single-threaded processes are never blocked.

1315 ▶ **Lemma 45.** *Let P be an arbitrary process:*

- 1316 (1) $iA(P) \cap \mathcal{L} \subseteq iA^\tau(P) \subseteq iA^*(P)$.
1317 (2) *If $P \equiv Q$ then $iA^*(P) = iA^*(Q)$.*
1318 (3) *If $P \xrightarrow{\alpha} Q$ for $\alpha \in \mathcal{R} \cup \{\tau\}$, then $iA^*(Q) \subseteq iA^*(P)$.*

1319 **Proof.** All points follow by an inspection of the SOS rules, using the definition of $iA^\tau(P)$
1320 and $iA^*(P)$. ◀

1321 On Interference

1322 ▶ **Proposition 46.** *Suppose $\alpha_1:H_1[E_1|R]$ and $\alpha_2:H_2[E_2|R]$ are interference-free. Then,*
1323 *$\alpha_1:H'_1[E_1]$ and $\alpha_2:H'_2[E_2]$ are interference-free for arbitrary subsets $H'_1 \subseteq H_1$ and $H'_2 \subseteq H_2$.*
1324

1325 As a special case, choosing $R = 0$, Proposition 46 implies non-interference of $\alpha_i:H'_i[E_i]$ for
1326 $i \in \{1, 2\}$ from non-interference of $\alpha_i:H_i[E_i]$ for any $H'_i \subseteq H_i$.

1327 **Proof.** First, non-interference of $\alpha_i:H_i[E_i|R]$ means $\alpha_i \notin H_{3-i}$ by assumption. Since
1328 $H'_i \subseteq H_i$, this implies also $\alpha_i \notin H'_{3-i}$. Second, suppose that $\alpha_{3-i} = \tau$ for some $i \in \{1, 2\}$. Then,
1329 non-interference of $\alpha_1:H_1[E_1|R]$ and $\alpha_2:H_2[E_2|R]$ gives $\tau \notin H_i$ and $H_i \cap \bar{iA}^*(E_i|R) = \{\}$.
1330 Since by Lem. 44, $\bar{iA}^*(E_i) \subseteq \bar{iA}^*(E_i|R)$, the inclusions $H'_i \subseteq H_i$ preserve this property, i.e.,
1331 $\tau \notin H'_i$ and $H'_i \cap \bar{iA}^*(E_i) \subseteq H_i \cap \bar{iA}^*(E_i|R) = \{\}$. ◀

1332 On Policies

1333 We obtain a partial ordering $\pi_1 \leq \pi_2$ on precedence policies by subset inclusion. Specifically,
1334 we have $(L_1, \dashrightarrow_1) \leq (L_2, \dashrightarrow_2)$ 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
1335 $\ell_1 \dashrightarrow_1 \ell_2$. Intuitively, if $\pi_1 \leq \pi_2$ then π_2 is a tightening of π_1 in the sense that it exports more
1336 labels (resources) subject to possibly fewer precedences (causality constraints) than π_1 . By
1337 way of this ordering, we can use policies to measure the degree of concurrency exhibited by a
1338 process. The alphabet of the \leq -maximal policy π_{max} contains all labels \mathcal{L} and its precedence
1339 relation \dashrightarrow_{max} is empty. It is the largest policy in the \leq ordering. The \leq -minimal policy
1340 π_{min} has empty alphabet and empty precedences. It is self-dual, i.e., $\bar{\pi}_{min} = \pi_{min}$.

1341 There is also the normal inclusion ordering $\pi_1 \subseteq \pi_2$ between policies defined by $L_1 \subseteq L_2$
1342 and $\dashrightarrow_1 \subseteq \dashrightarrow_2$. It differs from \leq in the inclusion direction of the precedence alphabets.

1343 For any policy π on alphabet L we define its *dual* as a policy $\bar{\pi}$ on the set of labels
1344 \bar{L} where for all $\bar{\ell}_1, \bar{\ell}_2 \in \bar{L}$ we stipulate $\bar{\ell}_1 \dashrightarrow \bar{\ell}_2 \in \bar{\pi}$ iff $\ell_1 = \ell_2$ or $\ell_1 \diamond \ell_2 \in \pi$, i.e., both
1345 $\ell_1 \dashrightarrow \ell_2 \notin \pi$ and $\ell_2 \dashrightarrow \ell_1 \notin \pi$. Intuitively, the dual policy $\bar{\pi}$ consists of matching copies $\bar{\ell}$ of
1346 all labels ℓ from π and puts them in a precedence relation precisely if their original copies

XX:38 Strong Priority and Determinacy in Timed CCS

1347 are identical or do not stand in precedence to each other, in any direction. This generates
 1348 a form of “negation” $\bar{\pi}$ from π , which is reflexive and symmetric. Note that $\bar{\bar{\pi}}$ need not be
 1349 identical to π but contains more precedences. In general, we have $\bar{\bar{\pi}} \leq \pi$ and $\bar{\bar{\pi}} = \pi$ if π is
 1350 symmetric. The policy $\bar{\pi}$ captures the independences between actions of π and is called the
 1351 associated *independence alphabet*, in the sense of trace theory [12]. Analogously, $\bar{\pi}$ expresses
 1352 the dependencies of actions in π , called the associated *dependence alphabet*.

1353 ► **Example 47.** Suppose $P \Vdash \ell_1 \diamond \ell_2$ then all transitions with labels ℓ_1 and ℓ_2 of P must be
 1354 from concurrent threads and thus cannot block each other. For instance, both $\ell_1 | \ell_2$ and
 1355 $\ell_1 + \ell_2$ conform to $\ell_1 \diamond \ell_2$. However, the choice $\ell_1 + \ell_2$ is not coherent. It becomes coherent
 1356 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,
 1357 we now have $\ell_1 + \ell_2 : \ell_1 \Vdash \ell_1 \dashrightarrow \ell_2$, where the policy expresses the precedence coded in the
 1358 blocking sets of the prefixes. Suppose $P \Vdash \ell_1 \diamond \ell_2$ then all transitions with labels ℓ_1 and ℓ_2
 1359 of P must be from concurrent threads and thus cannot block each other. For instance, both
 1360 $\ell_1 | \ell_2$ and $\ell_1 + \ell_2$ conform to $\ell_1 \diamond \ell_2$. However, the choice $\ell_1 + \ell_2$ is not coherent. It becomes
 1361 coherent if we add a precedence $\ell_1 + \ell_2 : \ell_1$ but then it is no longer conformant to $\ell_1 \diamond \ell_2$.
 1362 Instead, we now have $\ell_1 + \ell_2 : \ell_1 \Vdash \ell_1 \dashrightarrow \ell_2$, where the policy expresses the precedence coded
 1363 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,
 1364 then $P \Vdash \pi$ may take ℓ_1 and ℓ_2 transitions to completely different states from the same
 1365 thread.

1366 ► **Definition 48 (Pivot Policy).** A policy π is called a *pivot policy* if $\bar{\pi} \leq \pi$. We call a process
 1367 P *pivotable* if P conforms to a pivot policy.

1368 ► **Proposition 49.** A policy $\pi = (L, \dashrightarrow)$ is a *pivot policy* if $\bar{L} \subseteq L$ and for all $\ell_1, \ell_2 \in L$
 1369 with $\ell_1 \neq \ell_2$ we have $\ell_1 \diamond \ell_2 \in \pi$ or $\bar{\ell}_1 \diamond \bar{\ell}_2 \in \pi$.

1370 **Proof.** Let π be a pivot policy. Since the alphabet of $\bar{\pi}$ is \bar{L} and the alphabet of π is L the
 1371 assumption $\bar{\pi} \leq \pi$ implies $\bar{L} \subseteq L$ by definition of \leq . Given labels $\ell_1, \ell_2 \in L$ with $\ell_1 \neq \ell_2$
 1372 suppose $\ell_1 \diamond \ell_2 \notin \pi$. Hence $\ell_1 \dashrightarrow \bar{\ell}_2 \in \pi$ or $\ell_2 \dashrightarrow \bar{\ell}_1 \in \pi$. Consider the first case, i.e.,
 1373 $\ell_1 \dashrightarrow \bar{\ell}_2 \in \pi$. Since $\bar{L} \subseteq L$ and $L = \bar{\bar{L}}$ it follows that also $L \subseteq \bar{L}$, i.e., the labels ℓ_i are also
 1374 in the alphabet of $\bar{\pi}$. But then by definition of the ordering and the fact that $\ell_i = \bar{\bar{\ell}}_i$, the
 1375 assumption $\bar{\pi} \leq \pi$ implies that $\bar{\ell}_1 \dashrightarrow \bar{\ell}_2 \in \bar{\pi}$. Now, the definition of the dual policy means
 1376 that $\bar{\ell}_1 \dashrightarrow \bar{\ell}_2 \notin \pi$ and $\bar{\ell}_2 \dashrightarrow \bar{\ell}_1 \notin \pi$. This is the same as $\bar{\ell}_1 \diamond \bar{\ell}_2 \in \pi$ as desired.

1377 Suppose the properties (1) and (2) of the proposition hold for a policy π . We claim
 1378 that $\bar{\pi} \leq \pi$. The inclusion of alphabets is by assumption (1). Then, suppose $\bar{\ell}_1 \dashrightarrow \bar{\ell}_2 \in \pi$
 1379 which implies $\bar{\ell}_1 \diamond \bar{\ell}_2 \notin \pi$. If $\ell_1 = \ell_2$ we have $\bar{\ell}_1 \dashrightarrow \bar{\ell}_2 \in \bar{\pi}$ directly by definition. So, let
 1380 $\ell_1 \neq \ell_2$. Because of $\bar{L} \subseteq L$ and (2) this gives us $\ell_1 \diamond \ell_2 \in \pi$ considering again that $\ell_i = \bar{\bar{\ell}}_i$.
 1381 But $\ell_1 \diamond \ell_2 \in \pi$ is the same as $\bar{\ell}_1 \dashrightarrow \bar{\ell}_2 \in \bar{\pi}$ by definition. ◀

1382 ► **Proposition 50.** Suppose P is input-scheduled $\pi_{i.s.}$ and $P \xrightarrow[\bar{R}]{\alpha} H Q$ for some α, H, R, Q . Then:

- 1383 1. If $\alpha \in \bar{A}$ then $H \subseteq \{\alpha, \tau\}$.
- 1384 2. If $\alpha \in \mathcal{A} \cup \mathcal{C} \cup \{\tau\}$ then $H \subseteq \mathcal{A} \cup \mathcal{C} \cup \{\tau\}$.

1385 **Proof.** Obvious by definition of conformance and the construction of $\pi_{i.s.}$. ◀

1386 The next crucial Lemma implies that two coherent threads cannot block each other if
 1387 they are conformant to the same pivot policy. The blocking of a reduction $\ell | \bar{\ell}$ must always
 1388 arise from a thread in the concurrent context that is not involved in the synchronisation.

1389 ► **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
 1390 that $P_1 \xrightarrow[\bar{R}_1]{\ell} H_1 P'_1$ and $P_2 \xrightarrow[\bar{R}_2]{\bar{\ell}} H_2 P'_2$. Then, $H_2 \cap \bar{iA}(R_1) = \{\}$ implies $H_2 \cap \bar{iA}(P_1) \subseteq \{\bar{\ell}\}$.

Proof. Let $\ell \in \mathcal{L}$ and the transitions be given as in the statement of the Lemma. Suppose $\alpha \in H_2$ and $\alpha \in \overline{iA}(P_1)$. We claim that $\alpha = \bar{\ell}$, whence $H_2 \cap \overline{iA}(P_1) \subseteq \{\bar{\ell}\}$. By contraposition let us assume that $\alpha \neq \bar{\ell}$, or equivalently $\bar{\alpha} \neq \ell$. Further, since $iA(P_1) \subseteq \mathcal{L}$ we also have $\alpha \neq \tau$. The assumption $\alpha \in H_2$ means $\alpha \dashrightarrow \bar{\ell} \in \pi$ by Def. 12(1) applied to P_2 . Since π is pivot we must thus have $\bar{\alpha} \diamond \ell \in \pi$, specifically $\bar{\alpha} \dashrightarrow \ell \notin \pi$, i.e., $\bar{\alpha} \notin H_1$, again by Def. 12(1), now applied to P_1 . But since $\bar{\alpha} \in iA(P_1)$, by Lem. 44, there is a transition $P_1 \xrightarrow{\bar{\alpha}}_E Q$. Since also $\ell \dashrightarrow \bar{\alpha} \notin \pi$, by pivot property, it follows $\ell \notin H$ by Def. 12(1) again for P_1 . Now both $\bar{\alpha} \notin H_1$ and $\ell \notin H$ mean that the c-actions $\ell:H_1[R_1]$ and $\bar{\alpha}:H[E]$ are interference-free (Def. 53). Since $\ell \neq \bar{\alpha}$ there must exist a reconvergent process R and transitions

$$P_1 \xrightarrow{\bar{\alpha}}_{E'} R \text{ and } Q \xrightarrow{\ell}_{R'_1} R \text{ with } R_1 \xrightarrow{\bar{\alpha}} R'_1 \text{ and } E \xrightarrow{\ell} E'.$$

1391 The latter transitions are strengthenings of the residuals $R_1 \rightsquigarrow_{\bar{\alpha}} R'_1$ and $E \rightsquigarrow_{\ell} E'$ that must
 1392 exist according to the statement of Coherence, because of $\{\ell, \bar{\alpha}\} \subseteq \mathcal{R}$ and $\ell \neq \bar{\alpha}$. However, the
 1393 transition $R_1 \xrightarrow{\bar{\alpha}} R'_1$ implies $\bar{\alpha} \in iA(R_1)$ which is impossible, because then $\alpha \in H_2 \cap \overline{iA}(R_1)$
 1394 while $H_2 \cap \overline{iA}(R_1) = \{\}$ by assumption. ◀

1395

1396 We can show that pivotable processes do not have internal races.

1397 ▶ **Lemma 52.** *Let $P \Vdash \pi$ be coherent for a pivot policy π . Then, for every reduction*
 1398 $P \xrightarrow{\tau}_E P'$, *if $H \cap \overline{iA}(E) = \{\}$ then $\tau \notin H$.*

Proof. Let P be coherent for pivotable π and a reduction $P \xrightarrow{\tau}_E P'$ such that $H \cap \overline{iA}(E) = \{\}$ be given. Suppose, by contraposition, $\tau \in H$. Since there are no action prefixes $\tau.P'$ in CCS^{spt} , the reduction must arise from a synchronisation of two threads P_1 and P_2 inside P , via rule (Com), i.e.,

$$\frac{P_1 \xrightarrow{\ell}_{R_1} P'_1 \quad P_2 \xrightarrow{\bar{\ell}}_{R_2} P'_2 \quad B = \{\ell | \bar{\ell} \mid H_2 \cap \overline{iA}(P_1) \not\subseteq \{\bar{\ell}\} \text{ or } H_1 \cap \overline{iA}(P_2) \not\subseteq \{\ell\}\}}{P_1 | P_2 \xrightarrow{\ell | \bar{\ell}}_{R_1 | R_2} P'_1 | P'_2} \text{ (Com)}$$

1399 for some $\ell \in \mathcal{L}$. The assumption $H \cap \overline{iA}(E) = \{\}$ for the top-level reduction implies that
 1400 $(H_1 \cup H_2) \cap \overline{iA}(R_1 | R_2) = \{\}$ at the point of (Com). Here we may assume, by induction
 1401 on the depth of the derivation, that $\tau \notin H_1 \cup H_2$. We must show that the synchronisation
 1402 action τ cannot enter into the blocking set by (Com), i.e., $\tau = \ell | \bar{\ell} \notin B$, i.e., which is
 1403 the same as $H_2 \cap \overline{iA}(P_1) \subseteq \{\bar{\ell}\}$ and $H_1 \cap \overline{iA}(P_2) \subseteq \{\ell\}$. The sub-expressions $P_1 \Vdash \pi'$ and
 1404 $P_2 \Vdash \pi'$ are also coherent for a common pivot policy π' . This policy may contain local
 1405 labels that are restricted or hidden in the outer process P . Since by assumption we have
 1406 $H_1 \cap \overline{iA}(R_1) \subseteq (H_1 \cup H_2) \cap \overline{iA}(R_1 | R_2) = \{\}$ and likewise $H_2 \cap \overline{iA}(R_2) = \{\}$, we can apply
 1407 Lem. 51 to show $H_2 \cap \overline{iA}(P_1) \subseteq \{\bar{\ell}\}$ and $H_1 \cap \overline{iA}(P_2) \subseteq \{\ell\}$. This means $B = \{\}$ and we are
 1408 done. ◀

1409 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

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

1410 First, we observe that both Q and R are coherent:

XX:40 Strong Priority and Determinacy in Timed CCS

- Process Q has two admissible transitions

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

1411 which both are self-interfering. Thus, reconvergence is required only between the two
 1412 transitions which is guaranteed, because they are from concurrent threads. This can
 1413 also be obtained from Thm. 24 and the pivot policy with $b \dashrightarrow b \in \pi$, $c \dashrightarrow c \in \pi$ and
 1414 $a \dashrightarrow c \in \pi$.

- Process R has two admissible transitions, the c -action $R \xrightarrow[0]{\bar{b}}_{\{\bar{a}\}} R_1$ and $R \xrightarrow[0]{\bar{c}}_{\{\bar{a}\}} R_2$. Each
 1415 interferes with itself but not with the other. Thus, coherence implies that R_1 and
 1416 R_2 reconverge. This is indeed the case, since we have $R_1 \xrightarrow[0]{\bar{c}}_{\{\bar{a}\}} 0$ and $R_2 \xrightarrow[0]{\bar{b}}_{\{\bar{a}\}} 0$, with
 1417

1418 commuting actions. The concurrent contexts are residual since $0 \rightsquigarrow 0$ and $0 \rightsquigarrow 0$. Since
 1419 all admissible transitions of R_1 (and R_2) are reflexive and interfere with each other, R_1
 1420 (and R_2) are coherent, too.

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

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

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

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

1421 where the silent τ enters the blocking set, because $\{a\} \cap i\bar{A}(\bar{a} + \bar{c}:\bar{a}) \not\subseteq \{c\}$. This is a race
 1422 computed in rule (*Com*). Thus, the reduction $c|\bar{c}$ of the original state $Q | R$ is not longer
 1423 c -enabled in the state $\underline{c}:a | R_1$. This breaks coherence.

1424 Finally, we notice that $Q | R$ is not pivotable. A common policy π for $Q | R$ must obviously
 1425 include the precedences $\bar{a} \dashrightarrow \bar{c} \in \pi$, as well as $a \dashrightarrow c \in \pi$. This is not pivot according to
 1426 Def. 48.

1427 On Milner' Confluence Class

1428 As we have noted above, CCS corresponds to the unlocked and (blocking) free processes of
 1429 CCS^{spt} (Prop. 29). For these processes, admissible transitions and strongly enabled transitions
 1430 coincide. The key result of Milner [27] (Chap. 11, Prop. 19) is that confluence is preserved
 1431 by inaction 0 , prefixing $\ell.P$ and *confluent composition* defined as $P |_L Q = (P | Q) \setminus L$ where
 1432 $\mathcal{L}(P) \cap \mathcal{L}(Q) = \{\}$ and $\bar{\mathcal{L}}(P) \cap \mathcal{L}(Q) \subseteq L \cup \bar{L}$. We will show how confluence for this fragment
 1433 of CCS processes follows from our more general results on coherence. From now on, for the
 1434 rest of this section, we assume that all processes are unlocked, i.e., each prefix $\ell:H.Q$ has
 1435 $H \cup \{\ell\} \subseteq \mathcal{A} \cup \bar{\mathcal{A}}$.

1436 Let us call a process P *discrete* if all prefixes occurring in P are of form $\ell:\{\ell\}.Q$, i.e.,
 1437 they are self-blocking. Under strong enabling, a discrete process behaves like a CCS process
 1438 with the restriction that each action ℓ only synchronises if there is a matching partner $\bar{\ell}$ in
 1439 a *unique* other parallel thread. If the matching partner is not unique, then the scheduling
 1440 blocks. For instance, $\ell:\ell|\bar{\ell}:\bar{\ell}|\bar{\ell}:\bar{\ell}$ blocks because there are two concurrent senders $\bar{\ell}$ matching
 1441 the receiver ℓ . Now consider the fragment CCS^{cc} of discrete processes built from inaction
 1442 0 , self-blocking prefixes $\ell:\{\ell\}.Q$ and confluent composition. Milner's result says that the
 1443 processes in CCS^{cc} are confluent under admissible scheduling.

1444 To emulate Milner's result in our setting, we consider the *sort* $\mathcal{L}(P)$ of a process P as the
 1445 admissible actions of a *discrete* policy π_P with only reflexive precedences. In other words,
 1446 $\ell_1 \dashv\vdash \ell_2 \in \pi_P$ iff $\ell_1 = \ell_2$. It is easy to see that a discrete process always conforms to π_P .
 1447 Also, discrete policies are always pivot policies (Def. 48) and dependency-closed for all label
 1448 sets (Def. 48), as one shows easily. Thus, discrete processes, which are coherent for discrete
 1449 policies, fulfill all conditions of the preservation laws for inaction 0 (Prop. 60), self-blocking
 1450 action prefix $\ell:\{\ell\}.Q$ (Prop. 61), parallel composition $P|Q$ (Prop. 64) and restriction $P \setminus L$
 1451 (Prop. 66). As a consequence and in particular, all processes in CCS^{cc} are coherent under
 1452 strong enabling by our results.

1453 Our final observation now is that, for Milner's fragment CCS^{cc} , the transition semantics
 1454 under admissible and strong enabling, as well as the notions of confluence and coherence,
 1455 coincide. Firstly, if P is discrete then in every c-action $\ell:H[E]$ executed by a derivative Q of P
 1456 we must have $H = \{\ell\}$. Thus, the coherence Def. 12(1) becomes redundant since it is always
 1457 satisfied. Further, any two c-actions $\ell_i:H_i[E_i]$ of transitions $Q \xrightarrow[\ell_1]{E_1} Q_1$ and $Q \xrightarrow[\ell_2]{E_2} Q_2$
 1458 for $\ell_1 \neq \ell_2$ or $Q_1 \neq Q_2$ are trivially interference-free. The confluent composition of Milner
 1459 has the further effect that for every c-action $\ell:H[E]$ we must have $\ell \notin \bar{iA}^*(E)$. This is
 1460 proven by induction on the structure of $P \in \text{CCS}^{\text{cc}}$. Further, the side condition of (*Com*)
 1461 becomes trivial, and thus generally $\tau \notin H$. This is a result of the side-conditions of the
 1462 confluent composition. Hence, for CCS^{cc} , strong enabling coincides with plain admissibility
 1463 CCS. Notice that the class of discrete processes and the class processes coherent for discrete
 1464 policies is larger than CCS^{CW} . In particular, we can explain coherence of broadcast actions
 1465 with repetitive prefixes. Hence, our results are more general even for the restricted class of
 1466 discrete behaviours.

1467 **D** Proofs

1468 First, we adopt a slightly more general form of interference, for general c-actions not just for
 1469 those that are constructively enabled, as in Def. 11.

1470 **► Definition 53.** *Two c-actions $\alpha_1:H_1[E_1]$ and $\alpha_2:H_2[E_2]$ are called interference-free if the*
 1471 *following holds for all $i \in \{1, 2\}$:*

- 1472 (1) *If $\alpha_1 \neq \alpha_2$ then $\alpha_i \notin H_{3-i}$.*
 1473 (2) *If $\alpha_{3-i} = \tau$ then $H_i \cap (\bar{iA}^*(E_i) \cup \{\tau\}) = \{\}$.*

1474 The extra condition (2) in Def. 53 says that for a c-action not to interfere with a silent
 1475 (and thus uncontrollable) action it must be c-enabled. Obviously, Def. 53 is a weakening of
 1476 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
 1477 interference-free according to Def. 11.

1478 **► Proposition 54.** *If a coherent process P offers enabled transitions on a clock σ and another*
 1479 *distinct action $\alpha \neq \sigma$ with $P \xrightarrow[\alpha]{R_1} P_1$ and $P \xrightarrow[\sigma]{R_2} P_2$ then $\alpha \in H_2$ or $\sigma \in H_1$. In particular,*
 1480 *if $\alpha = \tau$ then $\sigma \in H_1$.*

1481 **Proof.** Since $R_2 = 0$ by Lem. 44(1) and $iA(0) = \{\}$, we have $\alpha \notin iA(R_2)$ and thus α cannot
 1482 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
 1483 coherence Def. 12(2) implies that the c-actions $\alpha:H_1[R_1]$ and $\sigma:H_2[R_2]$ must interfere. Since
 1484 $\alpha \neq \sigma$ this implies $\alpha \in H_2$ or $\sigma \in H_1$, or $\alpha = \tau$ and $H_2 \cap (\bar{iA}^*(R_2) \cup \{\tau\}) \neq \{\}$. Since $R_2 = 0$
 1485 the latter reduces to $\alpha = \tau \in H_2$. The last part of the Proposition now follows from the fact
 1486 that the clock transition is enabled and thus $\tau \notin H_2$. ◀

XX:42 Strong Priority and Determinacy in Timed CCS

1487 **Of Thm. 14.** Let P be coherent and Q a derivative with c-enabled reductions $Q \xrightarrow{R_1} Q_1$ and
 1488 $Q \xrightarrow{R_2} Q_2$. If $Q_1 \equiv Q_2$ we are done immediately. Suppose $Q_1 \not\equiv Q_2$. Then c-enabling
 1489 implies $\tau \notin H_i$ as well as $H_i \cap \overline{iA}^*(R_i) = \{\}$, which implies $H_i \cap (\overline{iA}^*(R_i) \cup \{\tau\}) = \{\}$. But
 1490 then the c-actions $\tau:H_1[R_1]$ and $\tau:H_2[R_2]$ are interference-free, whence coherence Def. 12(2)
 1491 implies there exist H'_i and processes R'_i, Q' such that $Q_1 \xrightarrow{R'_2} Q'$ and $Q_2 \xrightarrow{R'_1} Q'$
 1492 and $R_1 \rightsquigarrow R'_1$ and $R_2 \rightsquigarrow R'_2$ with $H'_i \subseteq H_i$. Finally, observe that $\overline{iA}^*(R'_i) \subseteq \overline{iA}^*(R_i)$
 1493 by Lem. 45 from which we infer $H'_i \cap (\overline{iA}^*(R'_i) \cup \{\tau\}) \subseteq H_i \cap (\overline{iA}^*(R_i) \cup \{\tau\})$. Thus, the
 1494 reconverging reductions $Q_i \xrightarrow{R'_{3-i}} Q'$ are again c-enabled. This was to be shown. ◀

1495 **Of Prop. 15.** The first part of the statement follows directly from Def. 12(2): The c-actions
 1496 $\ell:H_i[R_i]$ are trivially interference-free. Now if $P_1 \not\equiv P_2$, then coherence gives us a strong
 1497 environment shifts implying $\ell \in iA(R_1)$. The second part is because by Lem. 44(1) we have
 1498 $R_i = 0$ and thus $iA(R_i) = \{\}$ when $\ell \in C$. ◀

1499 ▶ **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
 1500 are single-threaded, then every admissible transition of $P_1 | P_2$ is c-enabled.*

Proof. Consider an admissible transition of $P_1 | P_2$ 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
 $\tau:(H_1 \cup H_2 \cup B)[R_1 | R_2]$ with $R_1 | R_2 = 0$. Because of Lem. 51, the set

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

1501 must be empty. Thus, the reduction $\ell_1 | \ell_2$ is c-enabled, too. ◀

1502 The following, slightly extended version of Prop. 23 says that a pivotable process cannot
 1503 offer a clock action and exhibit another clock or a rendez-vous synchronisation at the same
 1504 time. This means that in this class of processes, clocks and reductions are sequentially
 1505 scheduled.

1506 ▶ **Proposition 56 (Sequential Schedule and Clock Maximal Progress).** *Suppose P is coherent
 1507 and pivotable and $\sigma \in iA(P)$. Then, for all $\ell \in \mathcal{L}$ with $\ell \neq \sigma$ we have $\ell \notin iA(P)$ or $\bar{\ell} \notin iA(P)$.
 1508 In particular, P is in normal form, i.e., there is no reduction $P \xrightarrow{\tau} P'$.*

1509 **Proof.** The proof proceeds by contradiction. Let $P \Vdash \pi$ for pivot policy π . Suppose $\ell \in \mathcal{L}$,
 1510 $\ell \neq \sigma$ and $P \xrightarrow{\sigma} P_1$ and $P \xrightarrow{\ell} P_2$ and $P \xrightarrow{\bar{\ell}} P_3$. By Prop. 54 we infer $\sigma \in N$ or $\ell \in H$,
 1511 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
 1512 same reason we have $\pi \not\Vdash \sigma \diamond \bar{\ell}$. However, this contradicts the pivot property of π , which
 1513 requires $\pi \Vdash \sigma \diamond \ell$ or $\pi \Vdash \sigma \diamond \bar{\ell}$. Finally, if $P \xrightarrow{\tau} P'$ then the reduction must arise from a
 1514 rendez-vous synchronisation inside P , i.e., there is $\ell \in \mathcal{R}$ with $\ell \in iA(P)$ and $\bar{\ell} \in iA(P)$. But
 1515 then $\sigma \in iA(P)$ is impossible as we have just seen. ◀

1516 The proof of Thm. 24 will be conducted in the following Secs. D.1–D.4. It proceeds by
 1517 induction on the structure of derivations in the SOS of CCS^{spt} , given in Fig. 2. We first show
 1518 that the indirections of syntactic transformations via the static laws of Fig. 1, built into the
 1519 SOS by the (*Struct*) rule, can in fact be eliminated in terms of a finite number of standard
 1520 SOS rules that do not refer to structural laws.

1521 ▶ **Definition 57.** *A process P is called 0-free if either $P = 0$ or all occurrences of 0 in P are
 1522 guarded by a prefix $\ell:H.R$.*

1523 ▶ **Proposition 58.** *Every process P is structurally congruent to a 0-free process Q .*

1524 **Proof.** By application of the laws $P \star 0 \equiv P$ and $0 \wr L \equiv 0$, where $\star \in \{ |, + \}$ and $\wr \in \{ \setminus, / \}$
1525 we can eliminate every occurrence of 0 in an process P that is not of the form $\alpha:H.0$. ◀

▶ **Proposition 59.** *If P is a 0-free process, and there is a derivation \mathcal{D} for $P \xrightarrow[R]{\alpha} Q$, 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) :*

$$\frac{P \xrightarrow[R]{\alpha} P'}{P + Q \xrightarrow[R]{\alpha} P'} \text{ (Sum}_1\text{)} \qquad \frac{Q \xrightarrow[R]{\alpha} Q'}{P + Q \xrightarrow[R]{\alpha} Q'} \text{ (Sum}_2\text{)}$$

$$\frac{P \xrightarrow[R]{\alpha} P' \quad \alpha \notin \mathcal{C}}{P | Q \xrightarrow[R|Q]{\alpha} P' | Q} \text{ (Par}_1\text{)} \qquad \frac{Q \xrightarrow[R]{\alpha} Q' \quad \alpha \notin \mathcal{C}}{P | Q \xrightarrow[R|Q]{\alpha} P | Q'} \text{ (Par}_2\text{)}$$

1526 *More specifically, there is a derivation \mathcal{D}' for $P \xrightarrow[R']{\alpha} Q'$ with $R' \equiv R$ and $Q' \equiv Q$.*

1527 **Proof Sketch.** By induction on the structure of processes one shows that the transitions
1528 generated by the SOS with $(Struct)$ can be simulated by the extended SOS without $(Struct)$.
1529 The argument is based on the following case analysis:

- 1530 ■ If $P = 0$ then no process structurally equivalent to P can generate any transition: If
1531 $P \equiv Q$ and $Q \xrightarrow[R]{\alpha} Q'$ then $P \neq 0$. Hence, the statement of the proposition holds trivially.
- 1532 ■ For prefixes we show that if $P \equiv \ell:H.Q$ and $P \xrightarrow[E']{\ell'} Q'$ then $\ell' = \ell$, $H' = H$, $E' \equiv 0$ and
1533 $Q' \equiv Q$.
- 1534 ■ For sums we observe that if $P \equiv P_1 + P_2$ and $P \xrightarrow[E]{\ell} Q$ then there exists $i \in \{1, 2\}$ such
1535 that $P_i \xrightarrow[E']{\ell} Q'$ such that $E' \equiv E$ and $Q' \equiv Q$.
- 1536 ■ 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
1537 transitions $P \xrightarrow[R]{\alpha} P'$ can be obtained from transitions of P_i and Q .

1538 ◀

1539 Note that the restriction in Prop. 59 is necessary. If P is not 0-free, say $P = 0 | Q$, then
1540 we can have $0 | Q \xrightarrow{\sigma} Q'$ because of $(Struct)$, because $0 | Q \equiv Q$ and $Q \xrightarrow{\sigma} Q'$, but without
1541 $(Struct)$ we could not derive $0 | Q \xrightarrow{\sigma} Q'$.

1542 Because of Prop. 58 we may assume our processes are 0-free. Because of Prop. 59 we
1543 may prove the preservation of coherence (for 0-free processes) by induction on the SOS
1544 rules without considering $(Struct)$, but including the symmetric rules (Par_i) and (Sum_i) for
1545 $i \in \{1, 2\}$.

1546 D.1 Stop and Prefixes

1547 ▶ **Proposition 60.** *The process 0 is coherent for any policy, i.e., $0 \Vdash \pi$ for all π .*

1548 **Proof.** The inactive process 0 is locally coherent for trivial reasons, simply because it does
1549 not offer any transitions at all. There is no SOS rule applicable to it in the $(Struct)$ -free
1550 setting. Note that in the system with $(Struct)$ rule, we would have to argue that no process
1551 structurally equivalent to 0 has any transitions. ◀

1552 ▶ **Proposition 61.** *Let process Q be coherent for π . Then, for every action $\ell \in \mathcal{R}$ the prefix
1553 expression $\ell:H.Q$ is coherent for π , if $\ell \in H \subseteq \{ \ell' \mid \ell' \dashrightarrow \ell \in \pi \}$.*

XX:44 Strong Priority and Determinacy in Timed CCS

1554 **Proof.** A prefix expression $P = \ell:H.Q$ generates only a single transition by rule (Act) , so
 1555 that the assumption

$$1556 \quad P \xrightarrow[E_1]{\alpha_1} Q_1 \text{ and } P \xrightarrow[E_2]{\alpha_2} Q_2 \quad (1)$$

1557 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
 1558 $H \subseteq \{\beta \mid \beta \dashrightarrow \alpha \in \pi\}$. Since $\alpha_i = \ell \in H = H_i$ and $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ nothing needs to be proved.
 1559 Finally, note that the only immediate derivative of $\alpha:H.Q$ is Q which is coherent for π by
 1560 assumption.

1561 Again, it is important to notice that the SOS with $(Struct)$ would not immediately warrant
 1562 the conclusion that both derivations (1) are by (Act) . This is because these transitions
 1563 could be the transitions of two syntactically distinct (but structurally congruent) expressions
 1564 $P_1 \equiv P \equiv P_2$. Instead, we would have to argue that if $P = \ell:H.Q \equiv P'$ and $P' \xrightarrow[E']{\alpha'} Q'$
 1565 then $\alpha' = \ell$, $H' = H$, $E' \equiv 0$ and $Q' \equiv Q$. That this is the case is a property of our specific
 1566 system of structural laws and handled by the above Prop. 59. This would obviously be
 1567 violated, e.g., if we added a structural law like $\ell:H.Q \equiv \alpha':H'.Q'$ where $\alpha' \neq \ell$, $H' \neq H$,
 1568 $E' \neq 0$ or $Q' \neq Q$. ◀

1569 As Examples 8 and 9 show, a prefix $\ell:H.Q$ for $\ell \in \mathcal{R}$ can very well be coherent even if
 1570 $\ell \notin H$. Here we note that for clock prefixes, the blocking set is only constrained by the policy.

1571 ▶ **Proposition 62.** *If Q is coherent for π and $\sigma \in \mathcal{C}$ is a clock, then the prefix expression*
 1572 *$\sigma:H.Q$ is coherent for π , too, if $H \subseteq \{\beta \mid \beta \dashrightarrow \sigma \in \pi\}$.*

1573 **Proof.** The argument runs exactly as in case of Prop. 61. However, we do not need to
 1574 require condition $\sigma \in H$ because if $\alpha_i = \sigma$ then $\alpha_1 = \alpha_2$, $Q_1 \equiv Q_2$ and $\{\alpha_1, \alpha_2\} \not\subseteq \mathcal{R}$. So, no
 1575 reconvergence is required. Again, we observe that the only immediate derivative of $\sigma:H.Q$ is
 1576 Q which is coherent for π by assumption. ◀

1577 D.2 Summation

1578 ▶ **Proposition 63.** *Let $P_1 \Vdash \pi_1$ and $P_2 \Vdash \pi_2$ be coherent and for all pairs of initial transitions*
 1579 *$P \xrightarrow[F_1]{\alpha_1} P_1$ and $P \xrightarrow[F_2]{\alpha_2} P_2$ we have $\alpha_1 \neq \alpha_2$ as well as $\alpha_1 \in H_2$ or $\alpha_2 \in H_1$. Then $P_1 + P_2$*
 1580 *is coherent for π if $\pi_1 \subseteq \pi$ and $\pi_2 \subseteq \pi$.*

1581 **Proof.** Let $P = Q + R$ be given with Q and R locally coherent for π_1 and π_2 , respectively
 1582 and the rest as in the statement of the proposition. Every transition of P is either a
 1583 transition of Q or of R via rules (Sum_i) . Conformance to π directly follows from the same
 1584 property of Q or R , respectively. The proof of coherence Def. 12 is straightforward, exploiting
 1585 that two competing but non-interfering transitions must either be both from Q or both
 1586 from R . More precisely, suppose $P \xrightarrow[F_1]{\alpha_1} P_1$ and $P \xrightarrow[F_2]{\alpha_2} P_2$ with $\alpha_1 \neq \alpha_2$ or $P_1 \not\equiv P_2$ or
 1587 $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ 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-
 1588 free. Now if the two transitions of P are by (Sum_1) from Q or by (Sum_2) from R , then
 1589 $\alpha_1 \in iA(Q)$ and $\alpha_2 \in iA(R)$. But then by assumption, $\alpha_1 \neq \alpha_2$ and $\alpha_1 \in H_2$ or $\alpha_2 \in H_1$,
 1590 which implies that $\alpha_1:H_1[F_1]$ and $\alpha_2:H_2[F_2]$ are not interference-free. This means, to prove
 1591 coherence Def. 12(2) we may assume that both transitions of P are either by (Sum_1) from
 1592 Q or both by (Sum_2) from R . Suppose both reductions of $P = Q + R$ are generated by
 1593 rule (Sum_1) , i.e., $Q \xrightarrow[F_1]{\alpha_1} Q_1$ and $Q \xrightarrow[F_2]{\alpha_2} Q_2$ where the transitions generated by (Sum_1)
 1594 are $Q + R \xrightarrow[F_1]{\alpha_1} Q_1$ and $Q + R \xrightarrow[F_2]{\alpha_2} Q_2$. Under the given assumptions, the reconverging
 1595 transitions of coherence Def. 12(2) are obtained directly from coherence of Q , by induction.

1596 The case that both transitions are from R by rule (Sum_2) is symmetrical. Finally, the
 1597 immediate derivatives P_1 and P_2 of $P_1 + P_2$ are coherent for π , because they are coherent
 1598 for π_i by assumption and thus also for the common extension $\pi_i \subseteq \pi$. ◀

1599 D.3 Parallel

1600 ▶ **Lemma 64** (Key lemma). *Let $Q \Vdash \pi$ and $R \Vdash \pi$ be such that $\pi \setminus \mathcal{R}$ is a pivot policy. Then*
 1601 $(Q \mid R) \Vdash \pi$.

Proof. Let $P = Q \mid R$ such that both Q and R are coherent for π where $\pi \setminus \mathcal{R}$ is a pivot policy. For conformance consider a transition

$$P \xrightarrow[E]{\ell} P'$$

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)

$$Q \mid R \xrightarrow[0]{\sigma_{H_2 \cup N_2 \cup B_2}} Q_2 \mid R_2$$

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

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

1602 and $B_2 = \{\ell \mid \bar{\ell} \mid H_2 \cap \bar{iA}(R) \not\sqsubseteq \{\sigma\} \text{ or } N_2 \cap \bar{iA}(Q) \not\sqsubseteq \{\sigma\}\}$. By Lem. 51 we have $B_2 = \{\}$.
 1603 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
 1604 $\ell' \dashrightarrow \ell \in \pi$ 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 $Q \mid R$ are always parallel compositions $Q' \mid R'$ of derivatives of Q and R . Thus, we may assume that the immediate derivatives $Q' \mid R'$ are coherent for π because coherence of P and Q is preserved under transitions (i.e., by co-induction). Now suppose

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

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 $\ell \mid \bar{\ell}$ for the blocking set in (Com) altogether:

$$\frac{P \xrightarrow[R_1]{\ell} P' \quad Q \xrightarrow[R_2]{\bar{\ell}} Q'}{P \mid Q \xrightarrow[R_1 \mid R_2]{\ell \mid \bar{\ell}} P' \mid Q'} \text{ (Com)}$$

1605 We will use the name (Com_a) to refer to the instance of (Com) with $\ell \in \mathcal{R}$ and the name
 1606 (Com_σ) for the instance with $\ell \in \mathcal{C}$ and we proceed 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

XX:46 Strong Priority and Determinacy in Timed CCS

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 $\alpha_2 = \ell | \bar{\ell}$ and

$$Q | R \xrightarrow{F_1 | R}_{\alpha_1} Q_1 | R \text{ and } Q | R \xrightarrow{F_2 | G_2}_{\ell | \bar{\ell}} H'_2 \cup N_2 Q_2 | R_2$$

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

$$Q \xrightarrow{F_1}_{\alpha_1} Q_1 \text{ and } Q \xrightarrow{F_2}_{\ell} Q_2 \text{ and } R \xrightarrow{G_2}_{\bar{\ell}} N_2 R_2.$$

We may safely assume $\alpha_1 \neq \ell | \bar{\ell} = \tau$ or $Q_1 | R \neq Q_2 | R_2$. The third case $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ is excluded since $\alpha_2 = \tau$. Further, suppose the c-actions $\alpha_1 : H_1[F_1 | R]$ and $\ell | \bar{\ell} : (H'_2 \cup N_2)[F_2 | G_2]$ are non-interfering. Observe that $\{\alpha_1, \ell | \bar{\ell}\} \not\subseteq \mathcal{R}$ and also $\{\alpha_1, \ell | \bar{\ell}\} \cap C = \{\}$. 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 $H_1 \cap \bar{iA}^*(F_1 | R) = \{\}$. Since $\ell = \bar{\ell} \in \bar{iA}(R) \cap \mathcal{L} \subseteq \bar{iA}^*(F_1 | R)$ by Lem. 44 and Lem. 45, this implies that $\ell \notin H_1$ up front. Next, if $\alpha_1 \neq \ell | \bar{\ell}$, 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 $\alpha_1 = \ell | \bar{\ell} = \tau$, but then by non-interference Def. 11(2) we have the non-interference equation

$$(H'_2 \cup N_2) \cap (\bar{iA}^*(F_2 | G_2) \cup \{\tau\}) = H_2 \cap (\bar{iA}^*(E_2) \cup \{\tau\}) = \{\}$$

1607 from which $\alpha_1 = \tau \notin H'_2$ follows. This settles the first part of the non-interference
1608 property Def. 11(1) for the diverging transitions out of Q .

1609 For the second part of non-interference, suppose $\alpha_1 = \tau$. Then the non-interference
1610 assumption implies that $H'_2 \cap \bar{iA}^*(F_2 | G_2) = \{\}$. Now since $\bar{iA}^*(F_2) \subseteq \bar{iA}^*(F_2 | G_2)$ we
1611 conclude from this that $H'_2 \cap \bar{iA}^*(F_2) = \{\}$. This completes the proof that the c-actions
1612 $\alpha_1 : H_1[F_1]$ 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

$$Q_1 | R \xrightarrow{F'_2 | G_2}_{\ell | \bar{\ell}} H''_2 \cup N_2 Q' | R_2 \text{ and } Q_2 | R_2 \xrightarrow{F'_1 | R_2}_{\alpha_1} H'_1 Q' | R_2$$

1613 such that $H'_1 \subseteq H_1$ and $H''_2 \cup N_2 \subseteq H'_2 \cup N_2$. Regarding the concurrent environments, if
1614 $\alpha_1 \in \mathcal{R}$ then the coherence of Q guarantees that $F_1 \xrightarrow{\ell} F'_1$ and $F_2 \xrightarrow{\alpha_1} F'_2$ from which we
1615 infer $F_2 | G_2 \xrightarrow{\alpha_1} F'_2 | G_2$ and $F_1 | R \xrightarrow{\ell | \bar{\ell}} F'_1 | R_2$.

1616 If $\alpha_1 = \tau$ the above guarantee from the coherence of Q is weaker: We only have
1617 $F_1 \xrightarrow{\ell} F'_1$ and $F_2 \xrightarrow{\alpha_1} F'_2$. In any case we obtain the environment shifts $F_2 | G_2 \xrightarrow{\alpha_1} F'_2 | G_2$
1618 and $F_1 | R \xrightarrow{\ell | \bar{\ell}} F'_1 | R_2$, 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

$$Q \xrightarrow{F_1}_{\alpha_1} Q_1 \text{ and } R \xrightarrow{F_2}_{\alpha_2} R_2$$

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

$$Q | R \xrightarrow{F_1 | R}_{H_1}^{\alpha_1} Q_1 | R \text{ and } Q | R \xrightarrow{Q | F_2}_{H_2}^{\alpha_2} Q | R_2$$

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

$$Q_1 | R \xrightarrow{Q_1 | F_2}_{H_2}^{\alpha_2} Q_1 | R_2 \text{ and } Q | R_2 \xrightarrow{F_1 | R_2}_{H_1}^{\alpha_1} Q_1 | R_2.$$

1619 Obviously, $Q_1 | R_2$ reduces to $Q_1 | R_2$ and by (Par_1) and (Par_2) we also have the strong
1620 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
1621 for we have, in particular, $F_1 | R \xrightarrow{\alpha_2} F_1 | R_2$ and $Q | F_2 \xrightarrow{\alpha_1} Q_1 | F_2$. Notice that in this
1622 case we do not need to assume that Q or R are coherent, i.e., the assumption that the
1623 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,

$$Q | R \xrightarrow{F_1 | R}_{H_1}^{\alpha_1} Q_1 | R \text{ and } Q | R \xrightarrow{F_2 | R}_{H_2}^{\alpha_2} Q_2 | R$$

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

$$Q \xrightarrow{F_1}_{H_1}^{\alpha_1} Q_1 \text{ and } Q \xrightarrow{F_2}_{H_2}^{\alpha_2} Q_2$$

with $\alpha_1 \neq \alpha_2$ or $Q_1 | R \neq Q_2 | R$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_i \notin H_{3-i}$. This, in particular, implies $Q_1 \neq Q_2$. Further, assume non-interference of $\alpha_i: H_i[F_i | R]$. We obtain non-interference of $\alpha_i: H_i[F_i]$ by Prop. 46. We can therefore apply coherence Def. 12 of Q and obtain processes F'_1, F'_2 and Q' with reconverging transitions

$$Q_1 \xrightarrow{F'_2}_{H'_2}^{\alpha_2} Q' \text{ and } Q_2 \xrightarrow{F'_1}_{H'_1}^{\alpha_1} Q'$$

1624 with $H'_i \subseteq H_i$ and such that

$$1625 F_1 \xrightarrow{\alpha_2} F'_1 \text{ and } F_2 \xrightarrow{\alpha_1} F'_2 \tag{2}$$

1626 and if $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 \neq Q_2$, more strongly

$$1627 F_1 \xrightarrow{\alpha_2} F'_1 \text{ and } F_2 \xrightarrow{\alpha_1} F'_2. \tag{3}$$

Reapplying (Par_1) we construct the transitions

$$Q_1 | R \xrightarrow{F'_2 | R}_{H'_2}^{\alpha_2} Q' | R \text{ and } Q_2 | R \xrightarrow{F'_1 | R}_{H'_1}^{\alpha_1} Q' | R.$$

1628 Obviously, by (Par_1) and (Par_2) we also obtain transitions $F_1 | R \xrightarrow{\alpha_2} F'_1 | R$ and $F_2 | R \xrightarrow{\alpha_1} F'_2 | R$
1629 $F'_2 | R$ 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
1630 $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 | R \neq Q_2 | R$.

- $(Com_a) \diamond (Com_a)$: 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

$$Q | R \xrightarrow{E_i}_{H_i \cup N_i}^{\alpha_i} Q_i | R_i$$

1631 with $P_i = Q_i | R_i$ are τ -actions generated by the communication rule (Com) , i.e. $\alpha_1 =$
1632 $\ell_1 | \bar{\ell}_1 = \tau = \ell_2 | \bar{\ell}_2 = \alpha_2$ for actions $\ell_i \in \mathcal{R}$. Since $\alpha_1 = \alpha_2$ and $\{\alpha_1, \alpha_2\} = \{\tau\} \not\subseteq \mathcal{R}$ we

XX:48 Strong Priority and Determinacy in Timed CCS

1633 only need to prove confluence the case that $P_1 \neq P_2$, i.e., $Q_1 \neq Q_2$ or $R_1 \neq R_2$. Moreover,
 1634 we only need a weak environment shift, because $\{\alpha_1, \alpha_2\} = \{\tau\} \not\subseteq \mathcal{R}$ and $\{\alpha_1, \alpha_2\} \cap \mathcal{C} = \{\}$.
 1635 Also, non-interference Def. 11(2) means that $(H_i \cup N_i) \cap (\bar{i}\mathbf{A}^*(E_i) \cup \{\tau\}) = \{\}$, i.e.,

$$1636 \quad H_i \cap (\bar{i}\mathbf{A}^*(E_i) \cup \{\tau\}) = \{\} \text{ and } N_i \cap (\bar{i}\mathbf{A}^*(E_i) \cup \{\tau\}) = \{\}. \quad (4)$$

Thus, we are looking at transitions

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

1637 We claim that the two c-actions $\ell_i:H_i[F_i]$ of Q , and likewise the two c-actions $\bar{\ell}_i:N_i[G_i]$ of
 1638 R , are interference-free. Note that since $\ell_i \neq \tau$, we only need to consider the first part of
 1639 Def. 11(1). We first show that at least one of these pairs of c-actions must be interference
 1640 free, because of coherence and conformance to π where $\pi \setminus \mathcal{R}$ is a pivot policy. It will then
 1641 transpire that the other must be interference-free, too, because of the assumption (4).

1642 Obviously, if $\ell_1 = \ell_2$ then both $\ell_i:H_i[F_i]$ of Q and $\bar{\ell}_i:N_i[G_i]$ are interference-free for
 1643 trivial reasons. We distinguish two cases depending on whether ℓ_1 and ℓ_2 are identical or
 1644 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
 1645 a pivot policy. Thus, $\ell_1 \diamond \ell_2 \in \pi$ or $\bar{\ell}_1 \diamond \bar{\ell}_2 \in \pi$, by Def. 48. Hence, $\ell_i \notin H_{3-i}$ or $\bar{\ell}_i \notin N_{3-i}$
 1646 by conformance Def. 17, which means that at least one of the pairs of c-actions $\ell_i:H_i[F_i]$
 1647 of Q or the two c-actions $\bar{\ell}_i:N_i[G_i]$ 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 $\bar{\ell}_i:N_i[G_i]$ 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 $\ell_i \in i\mathbf{A}(F_{3-i}) \subseteq i\mathbf{A}(F_{3-i} | G_{3-i}) = i\mathbf{A}(E_{3-i}) \subseteq i\mathbf{A}^*(E_{3-i})$
 and therefore $\bar{\ell}_i \notin N_{3-i}$ because of (4). Thus, as claimed, the c-actions $\bar{\ell}_i:N_i[G_i]$ are
 interference-free, too. Again $\{\bar{\ell}_1, \bar{\ell}_2\} \subseteq \mathcal{R}$ and $\bar{\ell}_i \notin N_{3-i}$ 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) $G_1 \xrightarrow{\bar{\ell}_2} G'_1$ as well as $G_2 \xrightarrow{\bar{\ell}_1} G'_2$ as well as a process R' with

$$R_1 \xrightarrow[G'_2]{\bar{\ell}_2} N'_2 R' \text{ and } R_2 \xrightarrow[G'_1]{\bar{\ell}_1} N'_1 R'.$$

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

$$Q_1 | R_1 \xrightarrow[F'_2 | G'_2]{\ell_2 | \bar{\ell}_2} H'_2 \cup N'_2 Q' | R' \text{ and } Q_2 | R_2 \xrightarrow[F'_1 | G'_1]{\ell_1 | \bar{\ell}_1} H'_1 \cup N'_1 Q' | R'$$

with $H'_i \cup N'_i \subseteq H_i \cup N_i$, and also

$$F_i | G_i \xrightarrow{\ell_{3-i} | \bar{\ell}_{3-i}} F'_i | G'_i$$

1648 as required, since this implies the weaker form $F_i | G_i \rightsquigarrow F'_i | G'_i$ of environment shift.
 It remains to consider the case that $\ell_1 = \ell_2 = \tau$. By assumption $P_1 \neq P_2$ and so we
 must have $Q_1 \neq Q_2$ or $R_1 \neq R_2$. Suppose $Q_1 \neq Q_2$. If $R_1 \neq R_2$ we apply a symmetric
 argument with the role of ℓ_i and $\bar{\ell}_i$ 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 $Q_1 \neq Q_2$)

$$Q_1 \xrightarrow{F'_2}_{F'_2} Q' \text{ and } Q_2 \xrightarrow{F'_1}_{F'_1} Q'.$$

such that $H'_i \subseteq H_i$. This means that $\ell \in \text{iA}(F_{3-i}) \subseteq \text{iA}(E_{3-i})$ and so $\bar{\ell} \notin N_{3-i}$ because of $N_{3-i} \cap \overline{\text{iA}}^*(E_{3-i}) = \{\}$ as per (4). If $R_1 \equiv R_2$ then for each $i \in \{1, 2\}$, by coherence Def. 12 (as $\bar{\ell} \notin N_{3-i} \cup \mathcal{C}$) there is a transition

$$R_i \xrightarrow{G'_{3-i}}_{G'_{3-i}} R'$$

with $N'_i \subseteq N_i$ and $G_i \equiv G'_i$ or $G_i \xrightarrow{\bar{\ell}} G'_i$. Using these transitions we now recombine

$$Q_i | R_i \xrightarrow{F'_{3-i} | G'_{3-i}}_{F'_{3-i} | G'_{3-i}} Q' | R'$$

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 $G_i \xrightarrow{\bar{\ell}} G'_i$ then

$$F_i | G_i \xrightarrow{\ell | \bar{\ell}} F'_i | G'_i.$$

This means that, in all cases, $F_i | G_i \xrightarrow{\ell | \bar{\ell}} F'_i | G'_i$ as desired. If $R_1 \neq R_2$ 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 $\bar{\ell} \in \mathcal{R}$ and $R_1 \neq R_2$) $G_1 \xrightarrow{\ell} G'_1$ and $G_2 \xrightarrow{\ell} G'_2$ as well as a process R' with

$$R_1 \xrightarrow{G'_2}_{G'_2} R' \text{ and } R_2 \xrightarrow{G'_1}_{G'_1} R'.$$

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

$$Q_1 | R_1 \xrightarrow{F'_2 | G'_2}_{F'_2 | G'_2} Q' | R' \text{ and } Q_2 | R_2 \xrightarrow{F'_1 | G'_1}_{F'_1 | G'_1} Q' | R'$$

1649 with $H'_i \cup N'_i \subseteq H_i \cup N_i$, and also $F_i | G_i \xrightarrow{\ell | \bar{\ell}} F'_i | G'_i$ which also implies $F_i | G_i \xrightarrow{\ell | \bar{\ell}} F'_i | G'_i$
1650 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

$$Q | R \xrightarrow{E_1}_{E_1} Q_1 | R_1 \text{ and } Q | R \xrightarrow{\sigma}_0 Q_2 | R_2$$

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

$$Q \xrightarrow{\ell}_{F_1} Q_1 \text{ and } R \xrightarrow{\bar{\ell}}_{G_1} R_1 \text{ and } Q \xrightarrow{\sigma}_{H_2} Q_2 \text{ and } R \xrightarrow{\sigma}_{N_2} R_2.$$

1651 We assume that the c-actions $(\ell | \bar{\ell}):(H_1 \cup N_1 \cup B_1)[E_1]$ and $\sigma:(H_2 \cup N_2 \cup B_2)[0]$ are
1652 non-interfering. Note that $\ell \neq \sigma$ and $\bar{\ell} \neq \sigma$. First, observe that by Lem. 51, $B_1 = \{\}$ = B_2 .
1653 Also, $\ell | \bar{\ell} = \tau \neq \sigma$ and so the non-interference assumption means that $\sigma \notin H_1 \cup N_1 \cup B_1$

XX:50 Strong Priority and Determinacy in Timed CCS

1654 as well as $\tau \notin H_2 \cup N_2 \cup B_2$. The other part of the non-interference, from Def. 11(1),
1655 does not provide any extra information.

1656 We claim that $\ell \in H_2$. By contradiction, if $\ell \notin H_2$ then, since $\sigma \notin H_1$, the two c-actions
1657 $\ell:H_1[F_1]$ and $\sigma:H_2[0]$ of Q would be interference-free according to Def. 11. But then,
1658 since $\{\ell, \sigma\} \cap \mathcal{C} \neq \{\}$, coherence Def. 12 applied to Q would imply $\sigma \in \text{iA}(F_1)$ and $\ell \in \text{iA}(0)$.
1659 The latter, however, is impossible. Thus, as claimed, $\ell \in H_2$. Likewise, from $\sigma \notin N_1$ we
1660 derive $\bar{\ell} \in N_2$ in the same fashion.

1661 Now we invoke the fact that $\tau \notin B_2$, i.e. $H_2 \cap \overline{\text{iA}}(R) \subseteq \{\sigma\}$ and $N_2 \cap \overline{\text{iA}}(Q) \subseteq \{\sigma\}$. But
1662 $\bar{\ell} \in \text{iA}(R)$ and $\ell \in \text{iA}(Q)$, whence we must have $\ell \notin H_2$ and $\bar{\ell} \notin N_2$. This is a contradiction.
1663 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

$$Q | R \xrightarrow{F_1 | R}_{\alpha_1} Q_1 | R \text{ and } Q | R \xrightarrow{0}_{H_2 \cup N_2 \cup B_2} Q_2 | R_2$$

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

$$Q \xrightarrow{F_1}_{\alpha_1} Q_1 \text{ and } Q \xrightarrow{0}_{H_2} Q_2 \text{ and } R \xrightarrow{0}_{N_2} R_2.$$

1664 where $\alpha_1 \neq \sigma$ and the c-actions $\alpha_1:H_1[F_1 | R]$ and $\sigma:(H_2 \cup N_2 \cup B_2)[0]$ are interference-
1665 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
1666 $(H_2 \cup N_2 \cup B_2) \cap (\text{iA}^*(0) \cup \{\tau\}) = \{\}$, which in particular means $H_2 \cap (\text{iA}^*(0) \cup \{\tau\}) =$
1667 $\{\}$. Therefore, the c-actions $\alpha_1:H_1[F_1]$ and $\sigma:H_2[0]$ are interference-free. Note that
1668 $\{\alpha_1, \sigma\} \cap \mathcal{C} \neq \{\}$. Hence we can exploit coherence Def. 12 for Q , obtaining the stronger
1669 form of environment shift. But this would imply $\alpha_1 \in \text{iA}(0)$ which is impossible. So we
1670 find that transitions obtained by rules (Par_i) and (Com_σ) are never interference-free in
1671 the sense of coherence.

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

$$Q | R \xrightarrow{0}_{H_1 \cup N_1 \cup B_1} Q_1 | R_1 \text{ and } Q | R \xrightarrow{0}_{H_2 \cup N_2 \cup B_2} Q_2 | R_2$$

1672 with $P_i = Q_i | R_i$ and $E_i = F_i | G_i$ are generated by the communication rules (Com_σ) .
1673 Observing that $\{\sigma_1, \sigma_2\} \subseteq \mathcal{R}$ is impossible, here, we may assume $\sigma_1 \neq \sigma_2$ or $Q_1 | R_1 \neq$
1674 $Q_2 | R_2$. This can only hold true if $Q_1 \neq Q_2$ or $R_1 \neq R_2$. Furthermore, the two clock
1675 transitions of $Q_1 | Q_2$ can be assumed to be interference-free. From this it follows that
1676 the two underlying clock transitions $Q \xrightarrow{0}_{H_1} Q_1$ and $Q \xrightarrow{0}_{H_2} Q_2$ and the likewise the
1677 transitions $R \xrightarrow{0}_{N_1} R_1$ and $R \xrightarrow{0}_{N_2} R_2$ are interference-free. If $Q_1 \neq Q_2$ then we could
1678 apply coherence of Q and obtain $\sigma \in \text{iA}(0)$, if $R_1 \neq R_2$ the same follows from coherence
1679 of R . This is impossible, whence the case can be excluded. ◀

1681 D.4 Hiding

1682 ▶ **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*
1683 *is coherent for π .*

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

$$\frac{P \xrightarrow{E}_{\alpha} Q \quad H' = H - \{\sigma\}}{P/\sigma \xrightarrow{E/\sigma}_{\alpha/\sigma} Q/\sigma} \text{ (Hide)}$$

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

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

arise by (*Hide*) from transitions

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

1684 with $E_i = F_i/\sigma$, $P_i = Q_i/\sigma$ and $H_i = N_i - \{\sigma\}$. For coherence Def. 12 we assume that
 1685 $\alpha_1/\sigma \neq \alpha_2/\sigma$ 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
 1686 c-actions $\alpha_i:H_i[E_i]$ be interference-free. Observe that the first case implies $\alpha_1 \neq \alpha_2$ and the
 1687 second case means $Q_1 \not\equiv Q_2$. Moreover, because of the assumption that $\sigma \dashrightarrow \alpha_i \notin \pi$ we must
 1688 have $\sigma \notin N_1 \cup N_2$. But this means $H_i = N_i$.

1689 First, suppose that $\alpha_1 = \alpha_2 = \sigma$. But then the assumption $Q_1 \not\equiv Q_2$ contradicts strong
 1690 determinacy of clocks (Proposition 15). Thus, the actions α_i cannot both be the clock σ
 1691 that is hidden. The case $\alpha_i = \sigma$ and $\alpha_{3-i} \in Act - \{\sigma\}$, for some fixed $i \in \{1, 2\}$ can also
 1692 be excluded as follows: By Prop. 54 the clock transition and the non-clock transition must
 1693 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
 1694 of the above. The latter is outright impossible since then $\alpha_{3-i}/\sigma = \alpha_{3-i} \in N_i = H_i$ which
 1695 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

$$N_{3-i} \cap \overline{i\mathbf{A}^*}(F_{3-i}/\sigma) = H_{3-i} \cap \overline{i\mathbf{A}^*}(E_{3-i}) = \{ \}.$$

Since by assumption on policy conformance $\sigma \notin N_{3-i}$ the latter implies $N_{3-i} \cap \overline{i\mathbf{A}^*}(F_{3-i}) = \{ \}$. 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

$$Q_1 \xrightarrow[F'_2]{\alpha_2} N_2 Q' \text{ and } Q_2 \xrightarrow[F'_1]{\alpha_1} N_1 Q' \text{ with } F_1 \rightsquigarrow_{\alpha_2} F'_1 \text{ and } F_2 \rightsquigarrow_{\alpha_1} F'_2$$

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

$$F_1/\sigma \rightsquigarrow_{\alpha_2/\sigma} F'_1/\sigma \text{ and } F_2/\sigma \rightsquigarrow_{\alpha_1/\sigma} F'_2/\sigma.$$

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

$$F_1/\sigma \xrightarrow{\alpha_2/\sigma} F'_1/\sigma \text{ and } F_2/\sigma \xrightarrow{\alpha_1/\sigma} F'_2/\sigma.$$

1696 This completes the proof. ◀

1697 **D.5 Restriction**

1698 Let us next look at the restriction operator. The following definition is relevant for creating
1699 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 $Q = (s + a:s \mid b.\bar{s}) \setminus s$ which is conformant and pivotable. The signal s blocks the action a by priority in the thread $s + a:s$. For the communication partner \bar{s} to become active, however, it needs the synchronisation with an external action \bar{b} . The transition

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

1700 generated by $(Restr)$ does not provide enough information in the blocking set H and
1701 environment E for us to be able to characterise the environments in which the action a is
1702 blocked. For instance, the parallel composition $Q \mid \bar{b}$ will internally set the sender \bar{s} free and
1703 thus block the a -transition. From the above transition the (Com) rule permits $Q \mid \bar{b}$ to offer
1704 the a -step. This is not right as it is internally blocked and does not commute with the $b \mid \bar{b}$
1705 reduction. A simple solution to avoid such problems and preserve confluence is to force
1706 restricted signals so they do not block any visible actions. This restriction suffices in many
1707 cases. Another, more general solution is to use the extended rule $Restr^*$ discussed above
1708 after Example 34.

1709 **► Proposition 66.** *If $Q \Vdash \pi$ and π is precedence-closed for $L \cup \bar{L}$, 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:

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

Consider a transition

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

1710 with $H' = H - \{a, \bar{a}\}$, $\ell \notin \{a, \bar{a}\}$ and $\ell' \in H'$. Then $\ell' \notin \{a, \bar{a}\}$ and $\ell' \in H$. Conformance
1711 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
1712 distinct from a and \bar{a} . This ensures conformance.

If $P = Q \setminus a$ and the two non-interfering and diverging transitions

$$P_2 \xleftarrow[E_2]{\alpha_2} P \xrightarrow[E_1]{\alpha_1} P_1$$

arise from rule $(Restr)$ then we have diverging transitions

$$Q_2 \xleftarrow[F_2]{\alpha_2} Q \xrightarrow[F_1]{\alpha_1} Q_1$$

1713 with $H_i = N_i - \{a, \bar{a}\}$, $\alpha_i \notin \{a, \bar{a}\}$, where $E_i \equiv F_i \setminus a$ and $P_i \equiv Q_i \setminus a$. We assume $\alpha_1 \neq \alpha_2$
1714 or $P_1 \neq P_2$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_i \notin H_{3-i}$. Further, suppose non-interference of $\alpha_1: H_1[E_1]$
1715 and $\alpha_2: H_2[E_2]$. Note that in case $P_1 \neq P_2$ we immediately have $Q_1 \neq Q_2$. Next, notice
1716 that the premise of the $(Restr)$ rule ensures $\alpha_i \notin \{a, \bar{a}\}$. It is easy to see that this means
1717 $\{a, \bar{a}\} \cap N_i = \{ \}$, i.e., $H_i = N_i$. By contraposition, suppose $\ell \in \{a, \bar{a}\}$ and $\ell \in N_i$. Then
1718 $\ell \dashrightarrow \alpha_i \in \pi$. But π is precedence-closed for $\{a, \bar{a}\}$, so we would have $\alpha_i \in \{a, \bar{a}\}$ which is a
1719 contradiction. So, $H_i = N_i$.

All this means that to invoke coherence Def. 12 of the above Q is sufficient 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 $H_i \cap (\overline{iA^*}(E_i) \cup \{\tau\}) = \{\}$, which is the same as $\tau \notin H_i$ and the following set condition

$$(N_i - \{a, \bar{a}\}) \cap \overline{iA^*}(F_i \setminus a) = \{\}.$$

Since $\{a, \bar{a}\} \cap N_{3-i} = \{\}$ we have $N_i \cap \overline{iA^*}(F_i) \subseteq \mathcal{L} - \{a, \bar{a}\}$. We claim that the α_i -transitions of Q is enabled, specifically $N_i \cap \overline{iA^*}(F_i) = \{\}$ observing that $\tau \notin H_i = N_i$. By contraposition, suppose $\beta \in N_i \cap \overline{iA^*}(F_i) \subseteq \mathcal{L} - \{a, \bar{a}\}$. Hence, $\beta \notin \{a, \bar{a}\}$ and so $\beta \in N_i - \{a, \bar{a}\}$ and also $\beta \in \overline{iA^*}(F_i \setminus a)$, but this contradicts the above set-condition. Thus, we have shown that for all $i \in \{1, 2\}$, $\alpha_{3-i} = \tau$ implies $N_i \cap \overline{iA^*}(F_i) = \{\}$. 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

$$Q_1 \xrightarrow[F'_2]{\alpha_2} N'_2 Q' \text{ and } Q_2 \xrightarrow[F'_1]{\alpha_1} N'_1 Q' \text{ with } F_1 \xrightarrow{\alpha_2} F'_1 \text{ and } F_2 \xrightarrow{\alpha_1} F'_2$$

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

$$P_1 \xrightarrow[E'_2]{\alpha_2} H''_2 Q' \setminus a \text{ and } P_2 \xrightarrow[E'_1]{\alpha_1} H''_1 Q' \setminus a \text{ with } F_1 \setminus a \xrightarrow{\alpha_2} F'_1 \setminus a \text{ and } F_2 \setminus a \xrightarrow{\alpha_1} F'_2 \setminus a$$

1720 with $E'_i = F'_i \setminus a$ and $H''_i = N'_i - \{a, \bar{a}\} \subseteq N_i - \{a, \bar{a}\} = H_i$. In the special case that
 1721 $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ 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
 1722 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 $F_2 \setminus a \xrightarrow{\alpha_1} F'_2 \setminus a$.
 1723 This completes the proof. \blacktriangleleft