



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-04367635

HAL Id: hal-04367635

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

Submitted on 21 Feb 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 classical theory of process algebra with priorities, we identify a new scheduling mechanism, called (*sequentially*) *constructive reduction* which is designed to capture the essence of synchronous programming. The distinctive property of this evaluation strategy is to achieve determinism-by-construction for multi-cast concurrent communication. In particular, it permits us to model shared memory multi-threading with reaction to absence as it lies at the core of the programming language Esterel. In the technical setting of CCS extended by clocks and priorities, we prove for a large class of processes, which we call *structurally coherent* the confluence property for constructive reductions. We further show that under some syntactic restrictions, called *pivotable* the operators of prefix, summation, parallel composition, restriction and hiding preserve structural coherence. This covers a strictly larger class of processes compared to those that are confluent in Milner’s classical theory of CCS without priorities.

2012 ACM Subject Classification Theory of computation, Process algebras

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

Digital Object Identifier 10.4230/LIPIcs...

Funding *Luigi Liquori*: Partially funded by ETSI.

Michael Mendler: Partially funded by UniCA/I3S.

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

1 Introduction

Classical process algebras, like CCS [22], CSP [16], Π -calculus [24, 25], just to mention a few, have been devised as a model of concurrency where a system and its environment continuously interact to mutually control each other’s behaviour through synchronisation and communication actions. This contrasts with automata theory where the interaction is single-pass: The environment provides a sequence of input actions to be fully consumed by the automaton before it outputs a result. This result is a decision value, determining if the string is accepted or not. The difference is reflected in the semantical theory:

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

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

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



© Liquori and Mendler;

licensed under Creative Commons License CC-BY 4.0

Leibniz International Proceedings in Informatics

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

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

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

63 1.1 Resolving Nondeterminism

64 Concurrency, by expression or by implementation, is both a convenient and unavoidable
 65 feature of modern software systems. Functional determinism is crucial for maintaining
 66 predictability and to manage design complexity by simple mathematical models [19, 6, 10]. If
 67 all primitive data manipulations are deterministic (as in the λ -calculus), then non-determinism
 68 arises either from unpredictable input data or the unpredictable timing of communication
 69 actions. External non-determinism that is resolved by the input data is innocuous. A
 70 process that waits for input and then branches control depending on the value of the input
 71 is functionally predictable. If the input can arrive from different concurrent senders, then
 72 non-determinism arises if the external timing is unobservable. For instance, the behaviour of
 73 a shared data object, in general, may depend on the order in which two method calls are
 74 executed on it. This creates non-determinism if the actions do not commute, i.e. if the object's
 75 state transitions are not *confluent*. The three standard ways to resolve data races and achieve
 76 confluence are *isolation*, *cloning* and *precedence*. Let us discuss these briefly. If actions α_1 and
 77 α_2 affect two concurrently independent parts of Q , say, $Q \stackrel{\text{def}}{=} \alpha_1.Q_1 \mid \alpha_2.Q_2$, then commutation
 78 is guaranteed. The parallel composition $P \mid Q \mid R$ ends up in configuration $P_1 \mid Q_1 \mid Q_2 \mid R_2$
 79 no matter which of P or R communicates first, and with which of the two concurrent
 80 parts of $\alpha_1.Q_1$ or $\alpha_2.Q_2$ of Q they synchronise with if α_1 and α_2 are indistinguishable.
 81 This is the method of *isolation* which is captured naturally by the parallel operator of
 82 process algebras. Another solution is to *clone* a resource unboundedly often, via a repetition
 83 operator. If $\alpha_1 = \alpha = \alpha_2$ and $Q \stackrel{\text{def}}{=} \alpha^*.Q'$ means the prefix α is repeated infinitely often,
 84 i.e., $Q \xrightarrow{\alpha} (Q' \mid \alpha^*.Q')$, then Q will offer α to both processes P and R independently by
 85 cloning the residual Q' . The parallel composition $P \mid Q \mid R$ will always end up in configuration
 86 $P_1 \mid Q' \mid Q' \mid Q \mid R_2$ when both P and R have finished their communication with Q . In all cases,
 87 the shared object Q does not really care about the ordering in which α_1 and α_2 are executed.
 88 Yet, what if the behaviour of a process Q must depend on the choice between α_1 and
 89 α_2 ? Then, the canonical solution is to enforce a *scheduling precedence* on the actions of the

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

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

- 93 ■ If Q is a store where $\alpha_1 = a?v$ is a data input and $\alpha_2 = a!w$ an output, we want the
94 input take *precedence* over the output to implement data-flow causality. Let us write
95 $Q \xrightarrow{\alpha_2}_{\{\alpha_1\}} Q_2$ to express that the output α_2 only synchronises with a matching $\bar{\alpha}_2$ action
96 if the environment E does not offer to engage in an input $\bar{\alpha}_1$ at the same time. In contrast,
97 the input action $Q \xrightarrow{\alpha_1}_{\{\}} Q_1$ is not blocked by the output α_2 .
- 98 ■ Sometimes an ordering does not make sense, say if $\alpha_1 = a?v$ and $\alpha_2 = a?w$ are inputs of
99 distinct values $v \neq w$ to be received by Q . Since these actions make Q take on distinct
100 continuation states $Q_1 \neq Q_2$, the environment E must be blocked as it attempts to engage
101 in both α_1 and α_2 , concurrently from P and from R . This can be modelled as a precedence
102 of α_1 over α_2 and α_2 over α_1 which generates transitions $Q_2 \xleftarrow{\alpha_2}_{\{\alpha_1\}} Q \xrightarrow{\alpha_1}_{\{\alpha_2\}} Q_1$.

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

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

XX:4 Strong Priority and Determinacy in Timed CCS

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

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

2 A Process Algebra with Clocks and Priorities

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

2.1 CCS^{spt} Syntax

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

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

169 Intuitively, 0 denotes the *inactive* process which stops all computations, including broadcasting
 170 synchronisation actions. The *action prefix* $\alpha.H.P$ denotes a process that offers to engage
 171 in the action α and then behave as P while the *blocking* set $H \subseteq \mathcal{L}$ lists all actions taking
 172 precedence over α . A prefix with $\alpha \in \mathcal{R}$ denotes a CCS-style rendez-vous action. In case
 173 $\alpha \in \mathcal{C}$, it denotes a CSP-style broadcast action that can communicate with all the surrounding
 174 processes sharing on the same clock only when the clock is universally *consumed*. The process
 175 $A \in \mathcal{I}$, where \mathcal{I} is a set of *process identifiers*, refers to a process whose behaviour is specified
 176 indirectly by a definitional equation $A \stackrel{\text{def}}{=} P$. We assume that in every restriction expression
 177 $P \setminus L$ the set $L \subseteq \mathcal{R}$ consists of rendez-vous actions, and in every hiding expression P / L we
 178 have $L \subseteq \mathcal{C}$. These *scoping* operators act as name binders that introduce local scopes.

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

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

189 In our concrete notation for expressions using the abstract syntax of processes, we will
 190 assume that the binary operator $+$ is right associative and taking higher binding power
 191 than the binary operator \mid . Thus, we write $P + Q + R \mid S$ instead of $(P + (Q + R)) \mid S$.
 192 Also, we assume that the unary operators of prefix, restriction and hiding have higher
 193 binding power than the binary operators. Hence, we write $P \setminus c + a.b.c.Q \mid R / \sigma$ instead of
 194 $((P \setminus c) + (a.(b.(c.Q)))) \mid (R / \sigma)$. The *sub-processes* of a P are all sub-expressions of P and
 195 (recursively) all sub-processes of Q for definitions $A \stackrel{\text{def}}{=} Q$ where A is a sub-expression of P .
 196 Sometimes it is useful to consider processes in a fragment of CCS^{spt} . Specifically, a process is
 197 called *unlocked* if P does not contain any clock prefix $\sigma:H.Q$ as a sub-process. A process
 198 is called *sequential* or *single-threaded* if it does not contain any parallel sub-process $Q \mid R$.
 199 P is *action-closed*, or simply *closed*, if all sub-processes $\ell:H.Q$ in P occur in the scope of
 200 a restriction or hiding operator that binds ℓ . We write $\text{vA}(P)$ for the set of free ("visible")
 201 action names of P . We identify syntactically two processes that differ only in the choice of
 202 bound names, and we work under the *Barendregt's hygiene convention* [1].

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

213 **► Definition 1 (Structural Congruence \equiv).**

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

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

XX:6 Strong Priority and Determinacy in Timed CCS

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

■ **Figure 1** Structural Congruence.

$$\begin{array}{c}
\frac{}{\alpha:H.P \xrightarrow[0]{\alpha} P} (\text{Act}) \qquad \frac{P' \xrightarrow[R]{\alpha} Q' \quad P \equiv P' \quad Q \equiv Q' \quad R \equiv R'}{P \xrightarrow[R]{\alpha} Q} (\text{Struct}) \\
\frac{P \xrightarrow[R]{\alpha} Q \quad L' = L \cup \bar{L} \quad \alpha \notin L' \quad H' = H - L'}{P \setminus L \xrightarrow[R \setminus L]{\alpha} Q' \setminus L} (\text{Restr}) \qquad \frac{P \xrightarrow[R]{\alpha} Q \quad H' = H - L}{P / L \xrightarrow[R / L]{\alpha / L} Q' / L} (\text{Hide}) \\
\frac{P \xrightarrow[R]{\alpha} P' \quad \alpha \notin \mathcal{C}}{P | Q \xrightarrow[R | Q]{\alpha} P' | Q'} (\text{Par}_{1,2}) \qquad \frac{P \xrightarrow[R]{\alpha} P'}{P + Q \xrightarrow[R]{\alpha} P'} (\text{Sum}_{1,2}) \\
\frac{P \xrightarrow[R_1]{\ell} P' \quad Q \xrightarrow[R_2]{\bar{\ell}} Q' \quad H = \text{race}(P | Q)}{P | Q \xrightarrow[R_1 | R_2]{\ell | \bar{\ell}} P' | Q'} (\text{Par}_3) \\
\text{where} \quad \alpha / L \stackrel{\text{def}}{=} \begin{cases} \tau & \text{if } \alpha \in L \\ \alpha & \text{otherwise} \end{cases} \quad \ell | \bar{\ell} \stackrel{\text{def}}{=} \begin{cases} \tau & \text{if } \ell \in \mathcal{R} \\ \sigma & \text{if } \ell = \sigma \end{cases} \\
\text{race}(P | Q) \stackrel{\text{def}}{=} \begin{cases} \{\tau\} & \text{if } H_1 \cap \overline{\text{iA}}(Q) \not\subseteq \{\ell\} \text{ or } H_2 \cap \overline{\text{iA}}(P) \not\subseteq \{\bar{\ell}\} \\ \{\} & \text{otherwise} \end{cases}
\end{array}$$

■ **Figure 2** The CCS^{spt} Plotkin's Structural Operational Semantics.

2.2 Operational Semantics

226
227 The operational semantics for CCS^{spt} is given using a labelled transition relation $P \xrightarrow{\alpha} Q$
228 in the style of Plotkin's Structured Operational Semantics (SOS) [29, 28], where $\alpha \in \text{Act}$
229 is the action that labels the SOS transition. When $\alpha \in \mathcal{L}$ the transition corresponds to a
230 communication step, namely a rendez-vous action or a broadcasted clock action. When $\alpha = \tau$
231 is silent, then the transition corresponds to a *reduction* or an *evaluation* step. In addition, we
232 will decorate the nondeterministic SOS with annotations that provides sufficient information
233 to express a uniform and confluent reduction strategy based on scheduling precedences. We
234 will call this new reduction strategy (*sequentially constructive scheduling*). Including these
235 annotations, our SOS judgment takes the form $P \xrightarrow[R]{\alpha} Q$ where $P, Q, R \in \mathcal{P}$ are processes,
236 $\alpha \in \text{Act}$ is an action, and $H \subseteq \text{Act}$ a set of actions. We call the three composite labels $\alpha, H,$
237 and R a *c-action*, written $\alpha:H[R]$. The information provided by such a c-action is sufficient to
238 define our *constructive reduction* strategy for CCS^{spt} that satisfies a Church-Rosser property.

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

■ **Figure 3** Strong Initial Actions.

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

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

251 In rules of Fig. 2 the reader can observe how the annotation for the concurrent context
252 and the blocking sets are inductively built. In a nutshell: rule *(Act)* is the axiom annotating
253 in the transition the action and the (empty) environment. Rules *(Restr)* and *(Hide)* deal
254 with “local restriction” of rendez-vous actions and “local hiding” of broadcasted clocks. Rules
255 *(Par_{1,2})* and *(Sum_{1,2})* are almost standard and deal with parallelism and sum. Rule *(Struct)*
256 internalises in the transition the above presented structural equivalence of processes. Rule
257 *(Par₃)* is compositional as one can expect for both rendez-vous and broadcast, and it have
258 the extra *race* condition enriching the blocking set of the conclusion in case P can interfere
259 with Q and vice-versa. Note that if we ignore the scheduling labels, then the operational
260 semantics of unlocked processes coincides precisely with the operational semantics of CCS.
261 The rules for clock actions coincide with the semantics of CSP broadcast actions.

262 2.3 Constructive Scheduling

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

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

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

271 ▶ **Definition 3** (Weakly Enabled Transitions). *An admissible transition $P \xrightarrow{\alpha}_R Q$ is called*
272 *weakly enabled if $H \cap (\bar{iA}(R) \cup \{\tau\}) = \{\}$.*

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

XX:8 Strong Priority and Determinacy in Timed CCS

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

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

283 By looking at the definition, it is easy to see that $iA(P) \subseteq iA^\tau(P)$, so every strongly enabled
284 transition is weakly enabled: as such, for free processes, the notions of weak and strong
285 enabling coincide. Note also that enabling as a scheduling constraint is not monotonic under
286 parallel composition. The enabledness property of a transition is not, in general, preserved
287 when the process is placed into a concurrent context. If a reduction $P \xrightarrow[R]{\tau} P'$ is enabled,
288 then we know that $H \cap \overline{iA}(R) = \{\}$ or $H \cap \overline{iA}^\tau(R) = \{\}$, depending on what scheduling we
289 choose. This means that the context R cannot (weakly) generate a synchronisation with a
290 blocking initial action. This does not mean that an extension $R|Q$ may not perhaps have
291 $H \cap \overline{iA}(R|Q) \neq \{\}$ or $H \cap \overline{iA}^\tau(R|Q) \neq \{\}$. This happens if blocking actions come from Q
292 or, in the case of strong enabling, become reachable though the interactions of R and Q .
293 Then the parallel extension $P|Q$ does not permit a reduction $P|Q \xrightarrow[R]{\tau} P'|Q$.

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

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

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

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

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

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

326 **3 Examples**

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

330 ▶ **Example 7 (Read Before Write)**. Let us see how the rendez-vous synchronisation is scheduled
 331 by the precedences that come with a prioritised choice. Consider a “store” process $S \stackrel{\text{def}}{=} w+r:w$
 332 that offers a write action w and a read action r but prefers the write over the read. If we
 333 put S in parallel with a reader $R \stackrel{\text{def}}{=} \bar{r}$ we can let read actions r and \bar{r} synchronise in a
 334 reduction $S | R \xrightarrow{\tau}_{\{w\}} 0$. The fact that w takes priority over r is recorded in the blocking
 335 set $\{w\}$ which contains the action w . This will block R ’s transition when a parallel writer
 336 $W \stackrel{\text{def}}{=} \bar{w}$ is added. Specifically, the reduction $w+r:w | \bar{r} | \bar{w} \xrightarrow{\tau}_{\{w\}} \bar{w}$ is not weakly enabled,
 337 because $\{w\} \cap \bar{iA}(\bar{w}) = \{w\} \cap \{w\} \neq \{\}$. The only enabled reduction of process $S | R | W$
 338 is to let S and W synchronise to produce the transition $w+r:w | \bar{r} | \bar{w} \xrightarrow{\tau}_{\{w\}} \bar{r}$ which is even
 339 strongly enabled, since $\{w\} \cap \bar{iA}^*(\bar{r}) = \{\}$.

340 ▶ **Example 8 (Two Thread Process)**. The two-thread process $a:b+\bar{b} | \bar{a}+b:a$ has two admissible
 341 transitions under (Par_3) for $\tau.0$ through $a:b | \bar{a}$ or $\bar{b} | b:a$. However, since $b \in \bar{iA}(a:b+\bar{b})$ and
 342 $a \in \bar{iA}(\bar{a}+b:a)$ both the actions $a:b$ and $b:a$ are blocked.

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

348 ▶ **Example 9 (Deadlock)**. Consider the process $S \stackrel{\text{def}}{=} w_0+r_0:w_0 | w_1+r_1:w_1$ which offers
 349 receiver actions w_i and r_i for $i \in \{0, 1\}$ such that w_i has precedence over r_i . Now take processes
 350 $P_0 \stackrel{\text{def}}{=} \bar{r}_0.\bar{w}_1$ and $P_1 \stackrel{\text{def}}{=} \bar{r}_1.\bar{w}_0$ which first aim to synchronise with r_i and then sequentially
 351 afterwards with w_{1-i} . The parallel composition $P_0 | S | P_1$ has two initial τ -reductions
 352 $P_0 | w_0+r_0:w_0 | \bar{w}_0 \xleftarrow{\tau}_{P_0 | (w_0+r_0:w_0) \{w_1\}} P_0 | S | P_1 \xrightarrow{\tau}_{w_1+r_1:w_1 | P_1 \{w_0\}} \bar{w}_1 | w_1+r_1:w_1 | P_1$. These
 353 reductions are both weakly enabled. If we continue under weak enabling then the residual
 354 process $\bar{w}_1 | w_1+r_1:w_1 | P_1$ for the first reduction must first execute the synchronisation on w_1
 355 and thus proceed as $\bar{w}_1 | w_1+r_1:w_1 | P_1 \xrightarrow{\tau} P_1$ while the residual process $P_0 | w_0+r_0:w_0 | \bar{w}_0$
 356 can only reduce to P_0 . Hence, the resulting final outcome of executing $P_0 | S | P_1$ is non-
 357 deterministic. Observe that none of the two reductions displayed above is actually strongly
 358 enabled. For the right reduction we find that its concurrent environment $R_0 \stackrel{\text{def}}{=} w_1+r_1:w_1 | P_1$
 359 has a reduction sequence $R_0 \xrightarrow{\tau} \bar{w}_0$ and thus $\{w_0\} \cap \bar{iA}^*(R_0) \subseteq \{w_0\} \cap \bar{iA}^*(\bar{w}_0) = \{w_0\} \cap$
 360 $\{w_0\} \neq \{\}$. Symmetrically, for the concurrent environment $R_1 \stackrel{\text{def}}{=} P_0 | w_0+r_0:w_0$ of the left
 361 reduction we have $R_1 \xrightarrow{\tau} \bar{w}_1$ and hence $\{w_1\} \cap \bar{iA}^*(R_1) \neq \{\}$.

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

XX:10 Strong Priority and Determinacy in Timed CCS

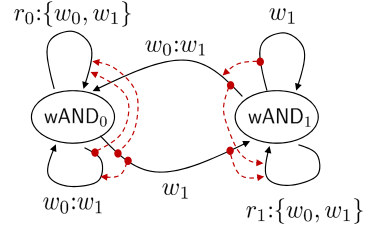
364 processes covered by CCS^{CW} [7] which restricts prioritised choice to receiver actions in the
 365 same way as the programming language Occam [2] with its “PRIALT” construct does.

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

$$368 \quad \text{wAND}_0 \stackrel{\text{def}}{=} w_1.\text{wAND}_1 + w_0:w_1.\text{wAND}_0 + r_0:\{w_0, w_1\}.\text{wAND}_0$$

$$369 \quad \text{wAND}_1 \stackrel{\text{def}}{=} w_1.\text{wAND}_1 + w_0:w_1.\text{wAND}_0 + r_1:\{w_0, w_1\}.\text{wAND}_1$$

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

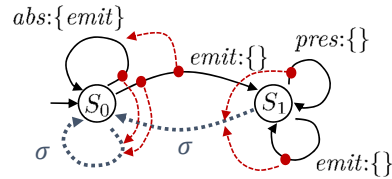


371 Between the write actions, the blocking sets implement a precedence of w_1 over w_0 . Notice
 372 that the blocking sets are visualised as dashed arrows.

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

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

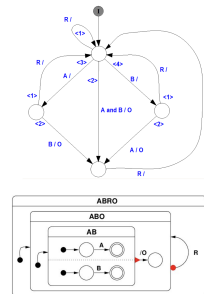
$$S_1 \stackrel{\text{def}}{=} \text{pres}.S_1 + \text{emit}.S_1 + \sigma:\{\text{pres}, \text{emit}\}.S_0$$



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

388 ▶ **Example 12** (ABRO). The *hallo word* in synchronous languages, see [4], ABRO has the
 389 following behaviour: *Emit the output o as soon as both inputs a and b have been received.*
 390 *Reset this behaviour each time the input r occurs, without emitting o in the reset cycle.* [32]

The behaviour of ABRO is expressed as a Mealy machine and as a SyncCharts, as displayed on the right. The synchronous behaviour of Esterel lies in the assumption that each transition of the Mealy machine summarises a single *macro-step* interaction of the machine with its environment. The transitions of parallel machines synchronise so that state changes proceed in lock-step. The simultaneous execution of these transitions that make up a macro-step consists of the sending and receiving of signals. The propagation of individual signals between machines is modeled in the *micro-step* operational semantics of Esterel.



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

■ **Figure 4** ABRO.

In CCS^{spt} we represent the micro-steps as sequences of τ -transitions, i.e., local and asynchronous rendez-vous synchronisations, whereas the macro-step is captured as a global clock synchronisation. Unfortunately, the Mealy machine formalism is not scalable because of the number of states can grow exponentially as the number of signals grow, while in Esterel can be expressed in a linear number of code lines as seen, as displayed on the right. The loop `loop P each R` of Esterel executes the behaviour of program `P` for an unbounded number of clock cycles, but restarts its body `P` when the signal `R` becomes present. The reset `R` takes priority over the behaviour of `P` which is preempted at the moment that the reset occurs. Note that in Esterel each signal appears exactly once, as in the specification and unlike in the Mealy machine. Other parts of Esterel also naturally arise from precedence-based scheduling. Esterel’s `loop P each R` mechanism, for instance, can be coded using the static parallel composition operator that combines the process `P` to be aborted and reset with a “watchdog” process `R`. The latter waits for the reset signal from the environment in each clock cycle. The choice between the reset and continuing inside `P` is resolved by giving `R` higher priority. When the reset occurs, the watchdog sends “kill” signals to all threads running in `P` and waits for these to terminate. Only then the watchdog `R` restarts `P` from scratch. This is a form of precedence that can be expressed in the blocking sets of CCS^{spt} . As such, we may present in the figure the following coding of the ABRO process in CCS^{spt} . The CCS^{spt} code follows the structure of the Esterel code. ABRO is obtained as a parallel composition `R | ABO` combining a watchdog `R` with the behaviour `ABO`. The `R` is responsible to implement the reset loop at the superstate called `ABO`, as in the state charts. The `ABO` is the behaviour of the interior of this superstate, which corresponds to the statement `(await A || await B); emit 0` in Esterel: it is the parallel composition of `A | B` making up the behaviour of the region called `AB` with code `await A || await B`. The processes `T | O` implement the synchronisation that waits for termination of `AB` to start the output process `O`. We assume that the channels `a, b, r, o` modelling interface signals `A, B, R, O` all have a unique sender and receiver (full-multicast in App. B.39). This is seen in our coding from the fact that the prefixes for these channels are self-blocking.

```

module ABRO:
input A, B, R;
output 0;
loop
await A || await B ;
emit 0
each R
end module

```

4 Constructive Scheduling for CCS^{spt}

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

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

XX:12 Strong Priority and Determinacy in Timed CCS

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

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

431 **► Definition 14.** Two *c-actions* $\alpha_1:H_1[E_1]$ and $\alpha_2:H_2[E_2]$ are called *interference-free* if the
 432 following holds for all $i \in \{1, 2\}$:

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

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

439 **► Proposition 15.** Suppose $\alpha_1:H_1[E_1 | R]$ and $\alpha_2:H_2[E_2 | R]$ are *interference-free*. Then,
 440 $\alpha_1:H'_1[E_1]$ and $\alpha_2:H'_2[E_2]$ are *interference-free* for arbitrary subsets $H'_1 \subseteq H_1$ and $H'_2 \subseteq H_2$.

441 Coherence is formulated using an auxiliary form of transition $P \rightsquigarrow^\alpha Q$ which abbreviates a
 442 form of *residual* transition with the following two cases: (i) If $\alpha \in \mathcal{L}$ then $P \xrightarrow{\alpha} Q$ or $P \equiv Q$
 443 and (ii) if $\alpha = \tau$ then $P \xrightarrow{\tau} Q$, $P \xrightarrow{\ell} Q$, or $P \xrightarrow{\bar{\ell}} Q$ for some $\ell \in \mathcal{R}$. This says that there is a
 444 transition from P to Q with an action that is a “factor” of α .
 445

446 **► Definition 16 (Coherence).** A process P is (structurally) *coherent* if for all its derivatives Q
 447 the following holds, where $\alpha_1, \alpha_2 \in \text{Act}$: Given two transitions $Q \xrightarrow[\mathcal{E}_1]{\alpha_1} Q_1$ and $Q \xrightarrow[\mathcal{E}_2]{\alpha_2} Q_2$
 448 such that $\alpha_1 \neq \alpha_2$ or $Q_1 \not\equiv Q_2$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_i \notin H_{3-i}$. Further, suppose the
 449 *c-actions* $\alpha_i:H_i[E_i]$ are *interference-free*. Then, there exist H'_i and processes E'_i, Q' such that
 450 $Q_1 \xrightarrow[\mathcal{E}'_2]{\alpha_2} Q'$ and $Q_2 \xrightarrow[\mathcal{E}'_1]{\alpha_1} Q'$ with $H'_i \subseteq H_i$ and $E_1 \rightsquigarrow^{\alpha_2} E'_1$ and $E_2 \rightsquigarrow^{\alpha_1} E'_2$. In particular,
 451 $E_1 \xrightarrow{\alpha_2} E'_1$ and $E_2 \xrightarrow{\alpha_1} E'_2$ if $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 \not\equiv Q_2$, or $\{\alpha_1, \alpha_2\} \cap \mathcal{C} \neq \{\}$.

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

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

457 **► Proposition 18.** If a coherent process P offers enabled transitions on a clock σ and another
 458 distinct action $\alpha \neq \sigma$ with $P \xrightarrow[\mathcal{R}_1]{\alpha} P_1$ and $P \xrightarrow[\mathcal{R}_2]{\sigma} P_2$ then $\alpha \in H_2$ or $\sigma \in H_1$. In particular,
 459 if $\alpha = \tau$ then $\sigma \in H_1$.

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

463 **► Proposition 19 (Action Determinism).** Let P be coherent with $P \xrightarrow[\mathcal{R}_1]{\ell} P_1$ such that $\ell \in \mathcal{L}$
 464 and $\ell \notin iA(\mathcal{R}_1)$. Then, for every transition $P \xrightarrow[\mathcal{R}_2]{\ell} P_2$ we have $P_1 \equiv P_2$. In particular, P is
 465 *structurally clock deterministic*, i.e., if $P \xrightarrow[\mathcal{R}_1]{\sigma} P_1$ and $P \xrightarrow[\mathcal{R}_2]{\sigma} P_2$ then $P_1 \equiv P_2$.

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

473 **► Definition 20 (Policy).** A precedence policy is a pair $\pi = (L, \dashrightarrow)$ where $L \subseteq \mathcal{L}$ is a subset
 474 of visible actions, called the alphabet of π and $\dashrightarrow \subseteq L \times L$ a binary relation on L , called
 475 the precedence relation. If $(\ell_1, \ell_2) \in \dashrightarrow$ we write $\pi \Vdash \ell_1 \dashrightarrow \ell_2$, or simply $\ell_1 \dashrightarrow \ell_2$ when π
 476 is understood. When $\ell \in L$ we say that ℓ is admissible under π and simply write $\ell \in \pi$. If
 477 $\ell_1 \dashrightarrow \ell_2$ we say that ℓ_1 takes precedence over ℓ_2 under π . If $\ell_1, \ell_2 \in L$ and $\pi \not\Vdash \ell_1 \dashrightarrow \ell_2$ as
 478 well as $\pi \not\Vdash \ell_2 \dashrightarrow \ell_1$ we say that ℓ_1 and ℓ_2 are concurrently independent under π , written
 479 $\pi \Vdash \ell_1 \diamond \ell_2$ or simply $\ell_1 \diamond \ell_2$ when π is understood.

480 We obtain a partial ordering $\pi_1 \leq \pi_2$ on precedence policies by subset inclusion. Specifically,
 481 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
 482 $\ell_1 \dashrightarrow_1 \ell_2$. Intuitively, if $\pi_1 \leq \pi_2$ then π_2 is a tightening of π_1 in the sense that it exports
 483 more labels (resources) subject to possibly fewer precedences (causality constraints) than
 484 π_1 . There is also the normal inclusion ordering $\pi_1 \subseteq \pi_2$ between policies defined by $L_1 \subseteq L_2$
 485 and $\dashrightarrow_1 \subseteq \dashrightarrow_2$. It differs from \leq in the inclusion direction of the precedence alphabets.
 486 Another notation we need is restriction. For any policy $\pi = (L, \dashrightarrow)$ and set of labels $K \subseteq \mathcal{L}$
 487 we denote by $\pi \setminus K$ the policy π restricted to the alphabet $L - (K \cup \overline{K})$ in the obvious way.

488 For any policy π on alphabet L we define its *dual* as a policy $\overline{\pi}$ on the set of labels
 489 \overline{L} where for all $\overline{\ell}_1, \overline{\ell}_2 \in \overline{L}$ we stipulate $\overline{\pi} \Vdash \overline{\ell}_1 \dashrightarrow \overline{\ell}_2$ iff $\ell_1 = \ell_2$ or $\pi \Vdash \ell_1 \diamond \ell_2$, i.e., both
 490 $\pi \not\Vdash \ell_1 \dashrightarrow \ell_2$ and $\pi \not\Vdash \ell_2 \dashrightarrow \ell_1$. Intuitively, the dual policy $\overline{\pi}$ consists of matching copies $\overline{\ell}$
 491 of all labels ℓ from π and puts them in a precedence relation precisely if their original copies
 492 are identical or do not stand in precedence to each other, in any direction. This generates
 493 a form of “negation” $\overline{\pi}$ from π , which is reflexive and symmetric. Note that $\overline{\overline{\pi}}$ need not be
 494 identical to π but contains more precedences. In general, we have $\overline{\overline{\pi}} \leq \pi$ and $\overline{\overline{\pi}} = \pi$ if π is
 495 symmetric. The policy $\overline{\pi}$ captures the independences between actions of π and is called the
 496 associated *independence alphabet*, in the sense of trace theory [9]. Analogously, $\overline{\pi}$ expresses
 497 the dependencies of actions in π , called the associated *dependence alphabet*.

498 We are going to use policies to measure the degree of concurrency exhibited by a process.
 499 The alphabet of the \leq -maximal policy π_{max} contains all labels \mathcal{L} and its precedence relation
 500 $\dashrightarrow_{\text{max}}$ is empty. It is the largest policy in the \leq ordering. Its dual $\overline{\pi}_{\text{max}}$ contains all labels
 501 and precedences between all of them, i.e., $\overline{\pi}_{\text{max}} \not\Vdash \ell_1 \dashrightarrow \ell_2$ for all $\ell_1, \ell_2 \in \mathcal{L}$. The \leq -minimal
 502 policy π_{min} has empty alphabet and empty precedences. It is self-dual, i.e., $\overline{\pi}_{\text{min}} = \pi_{\text{min}}$.
 503 The following definition explains how policies can statically specify the externally observable
 504 scheduling behaviour of a process. The maximal policy π_{max} is least restrictive in the sense
 505 that it permits a process to offer arbitrary actions with arbitrary blocking. The minimal
 506 policy π_{min} is most restrictive in that it does not permit any externally visible actions.

507 **► Definition 21 (Conformance).** P conforms to a policy π if for each of its derivatives Q : If
 508 $Q \xrightarrow[\text{R}]{\ell} Q'$ with $\ell \in \mathcal{L}$, then $\ell \in \pi$ and for all $\ell' \in \mathcal{L}$, if $\ell' \in H$ then $\pi \Vdash \ell' \dashrightarrow \ell$.

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

XX:14 Strong Priority and Determinacy in Timed CCS

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

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

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

524 **■** A policy π is a pivot policy if $\bar{\pi} \leq \pi$.

525 **■** A policy π is input-scheduled if $\pi \subseteq \pi_{is}$ where $\pi_{is} = (L_{is}, \dashrightarrow_{is})$ given by $L_{is} = \mathcal{L}$ and
 526 $\pi_{is} \Vdash \ell_1 \dashrightarrow \ell_2$ iff $\ell_1 = \ell_2$ or $\ell_1, \ell_2 \in \mathcal{A} \cup \mathcal{C}$.

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

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

530 In a pivotable process, the choice between any two synchronisations is resolved by the
 531 precedences in third independent thread, acting as the “pivot”. For instance, the process
 532 $P \stackrel{\text{def}}{=} a + \bar{b}:a \mid b + \bar{a}:b$ is not pivotable (see also Ex. 8 in App. B). It consists of two threads,
 533 each able to synchronise on a and also b . Which one is chosen, must be resolved between the
 534 two threads. This is not possible, because the first wants channel a to take priority over \bar{b}
 535 and the second the other way around. This is reflected in any associated policy π for which P
 536 is conformant, which must satisfy $\pi \Vdash a \dashrightarrow \bar{b}$ and $\pi \Vdash b \dashrightarrow \bar{a}$. Obviously, this is not a pivot
 537 policy which requires $\pi \Vdash a \diamond \bar{b}$ or $\pi \Vdash \bar{a} \diamond b$. For contrast, the process $Q \stackrel{\text{def}}{=} a + \bar{b}:a \mid \bar{a}$ is
 538 pivotable as it conforms to the pivot policy with $\pi \Vdash a \dashrightarrow \bar{b}$ and $\pi \Vdash \bar{a} \diamond b$. Here, the choice
 539 between synchronising on a or on b is uniquely resolved by the first thread $a + \bar{b}:a$ which,
 540 being offered \bar{a} and b concurrently by its environment, takes the synchronisation $\tau = a \mid \bar{a}$.

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

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

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

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

562 ▶ **Proposition 24** (Clock Maximal Progress). *Suppose P is coherent and pivotable and $\sigma \in$
 563 $iA(P)$. Then, for all $\ell \in \mathcal{L}$ with $\ell \neq \sigma$ we have $\ell \notin iA(P)$ or $\bar{\ell} \notin iA(P)$. In particular, P is in
 564 normal form, i.e., there is no reduction $P \xrightarrow{\tau} P'$.*

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

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

- 578 ■ $0 : \pi$ for all π .
- 579 ■ If $Q : \pi$ then for every $\ell \in \mathcal{R}$ we have $\ell:H.Q : \pi$ if $\ell \in H \subseteq \{\ell' \mid \pi \Vdash \ell' \dashrightarrow \ell\}$.
- 580 ■ If $Q : \pi$ and clock $\sigma \in \mathcal{C}$, then $\sigma:H.Q : \pi$ if $H \subseteq \{\beta \mid \pi \Vdash \beta \dashrightarrow \sigma\}$.
- 581 ■ If $P_1 : \pi_1$ and $P_2 : \pi_2$ and for all pairs of initial transitions $P \xrightarrow[\mathcal{F}_1]{\alpha_1} P_1$ and $P \xrightarrow[\mathcal{F}_2]{\alpha_2} P_2$
 582 we have $\alpha_1 \neq \alpha_2$ as well as $\alpha_1 \in H_2$ or $\alpha_2 \in H_1$. Then $P_1 + P_2 : \pi$ if $\pi_1 \subseteq \pi$ and $\pi_2 \subseteq \pi$.
- 583 ■ If $Q : \pi$ and $R : \pi$ where $\pi \setminus \mathcal{R}$ is a pivot policy. Then $(Q \mid R) : \pi$.
- 584 ■ If $P : \pi$ and $\pi \setminus \mathcal{R}$ is a pivot policy then for each $\ell \in \pi \setminus \mathcal{R}$, the sequential bang prefix
 585 $\ell^*:H.P : \pi$ if $H \subseteq \{\ell' \mid \pi \Vdash \ell' \dashrightarrow \ell\}$.
- 586 ■ If $Q : \pi$ and $\pi \not\vdash \sigma \dashrightarrow \ell$ for all $\sigma \in L$, then $Q / L : \pi$.
- 587 ■ If $Q : \pi$ and π precedence-closed for $L \cup \bar{L}$, then $(Q \setminus L) : \pi - L$.

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

590 **5 Further work**

591 We list in spare order some possible future works:

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

602 — **References** —

- 603 1 H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in*
604 *Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, revised edition, 1984.
- 605 2 G. Barrett. The semantics of priority and fairness in OCCAM. In M. Main, A. Melton,
606 M. Mislove, and D. Schmidt, editors, *Proc. MFPS'89*, 1989.
- 607 3 G. Berry. *The Constructive Semantics of Pure Esterel*. Draft Book, 1999. [ftp://ftp-sop-](ftp://ftp-sop.inria.fr/esterel/pub/papers/constructiveness3.ps)
608 [inria.fr/esterel/pub/papers/constructiveness3.ps](ftp://ftp-sop.inria.fr/esterel/pub/papers/constructiveness3.ps).
- 609 4 G. Berry. The Foundations of Esterel. In *Proof, Language and Interaction: Essays in Honour*
610 *of Robin Milner*, pages 425–454. MIT Press, 2000.
- 611 5 G. Berry and G. Boudol. The chemical abstract machine. *Theor. Comput. Sci.*,
612 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.
613 1016/0304-3975(92)90185-I.
- 614 6 R. Bocchino, V. Adve, S. Adve, and M. Snir. Parallel programming must be deterministic by
615 default. In *Proceedings of the First USENIX conference on Hot topics in parallelism*, pages
616 4–4, 2009.
- 617 7 J. Camilleri and G. Winskel. CCS with priority choice. *Information and Computation*,
618 116(1):26–37, 1995.
- 619 8 R. Cleaveland, G. Lüttgen, and M. Mendler. An algebraic theory of multiple clocks. In
620 A. Mazurkiewicz and J. Winkowski, editors, *Proceedings of the International Conference on*
621 *Concurrency Theory CONCUR'97*, pages 166–180. Springer, 1997. LNCS 1243.
- 622 9 V. Dieckert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.
- 623 10 S. A. Edwards. On determinism. In M. Lohstroh et. al., editor, *Lee Festschrift*, pages 240–253.
624 Springer, 2018.
- 625 11 P. Le Guernic, T. Goutier, M. Le Borgne, and C. Le Maire. Programming real time applications
626 with SIGNAL. In *Proceedings of the IEEE*, volume 79, pages 1321–1336. IEEE Press, September
627 1991.
- 628 12 M. Mendler H. R. Andersen. An asynchronous process algebra with multiple clocks. In
629 D. Sannella, editor, *In Proc. ESOP'94*, pages 58–73, 1994.
- 630 13 N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming
631 language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, September 1991.
- 632 14 D. Harel, A. Pnueli, J. Pruzan-Schmidt, and R. Sherman. On the formal semantics of
633 Statecharts. In *Logic in Computer Science (LICS'87)*, pages 54–64. IEEE Press, 1987.
- 634 15 M. Hennessy and T. Regan. A process algebra for timed system. *Information and Computation*,
635 117:221–239, 1995.
- 636 16 C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985. 256 pages.
- 637 17 C. D. Kloos and P. Breuer, editors. *Formal Semantics for VHDL*. Kluwer, 1995.
- 638 18 J. W. Klop. *Combinatory reduction systems*. PhD thesis, Univ. Utrecht, 1980.
- 639 19 E. A. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.
- 640 20 E. A. Lee and T. M. Parks. Dataflow process networks. In *Proceedings of the IEEE*, pages
641 773–799, May 1995.
- 642 21 G. Lüttgen and M. Mendler. When 1 clock is not enough. In L. Aceto and A.D. Gordon,
643 editors, *Algebraic Process Calculi: The First Twenty Five Years and Beyond (PA'05)*, number
644 NS-05-03 in BRICS Notes Series, pages 155–158, June 2005.
- 645 22 R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
- 646 23 R. Milner. Functions as processes. *Math. Struct. Comput. Sci.*, 2(2):119–141, 1992. URL:
647 <https://doi.org/10.1017/S0960129500001407>, doi:10.1017/S0960129500001407.
- 648 24 R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*,
649 100(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/
650 0890-5401(92)90008-4.
- 651 25 R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, II. *Inf. Comput.*,
652 100(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/
653 0890-5401(92)90009-5.

- 654 26 B. Norton, G. Luetzgen, and M. Mendler. A compositional semantic theory for component-
655 based synchronous design. In R. Amadio and D. Lugiez, editors, *Int'l Conf. on Concurrency*
656 *Theory (CONCUR'2003)*, pages 461–476, Marseille, September 2003. Springer LNCS 2761.
- 657 27 I. Phillips. CCS with priority guards. In K. G. Larsen and M. Nielsen, editors, *Proc. Concur*
658 *2001*, pages 305–320, 2001.
- 659 28 G. D. Plotkin. The origins of structural operational semantics. *J. Log. Algebraic Methods*
660 *Program.*, 60-61:3–15, 2004. URL: <https://doi.org/10.1016/j.jlap.2004.03.009>, doi:
661 10.1016/J.JLAP.2004.03.009.
- 662 29 G. D. Plotkin. A structural approach to operational semantics. *J. Log. Algebraic Methods*
663 *Program.*, 60-61:17–139, 2004.
- 664 30 V. Natarajan R. Cleaveland, G. Luetzgen. *Handbook of Process Algebra*, chapter Priority in
665 Process Algebra. Elsevier, 2001.
- 666 31 K. Schneider. The synchronous programming language Quartz. Internal Report 375,
667 Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany,
668 December 2009.
- 669 32 Esterel Technologies. The Esterel v7 reference manual version v7_30 – initial IEEE
670 standardization proposal. Technical report, Esterel Technologies, 679 av. Dr. J. Lefebvre,
671 06270 Villeneuve-Loubet, France, November 2005.
- 672 33 S.H. Unger. Asynchronous sequential switching circuits with unrestricted input changes. *IEEE*
673 *Transactions on Computers*, C-20(12):1437–1444, 1971. doi:10.1109/T-C.1971.223155.
- 674 34 L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111,
675 1990. URL: <https://doi.org/10.1145/79173.79181>, doi:10.1145/79173.79181.
- 676 35 R. J. van Glabbeek. Notes on the methodology of CCS and CSP. *Theor. Comput. Sci.*,
677 177(2):329–349, 1997. URL: [https://doi.org/10.1016/S0304-3975\(96\)00251-4](https://doi.org/10.1016/S0304-3975(96)00251-4), doi:10.
678 1016/S0304-3975(96)00251-4.
- 679 36 R. von Hanxleden, B. Duderstadt, C. Motika, S. Smyth, M. Mendler, J. Aguado, S. Mercer, and
680 O. O'Brien. SCCharts: Sequentially Constructive Statecharts for safety-critical applications.
681 In *Proc. PLDI'14*, Edinburgh, UK, June 2014. ACM.

682 **A** From Asynchronous Weak to Synchronous Strong Priorities

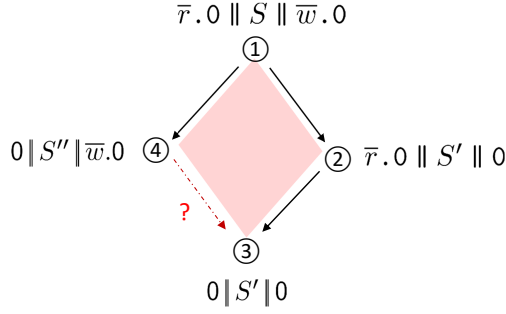
683 This rather informal section give to the curious reader a nice view of our personal pilgrimage
684 of previous work looking at reaching more control and determinism in asynchronous process
685 algebra. This trip lead us to consider adding clocks to the previous literature and reach
686 confluence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

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

688 Let us look in more detail on how the precedence mechanism suggested above might be
689 derived from the classical priorities studied for CCS. As a simple example, suppose S is a
690 (synchronous write, asynchronous read) memory cell that can be read at any time offering
691 the read action r , but only be written once with the write action w . Ignoring data values, the
692 cell would be modelled in plain CCS² by the equations $S \stackrel{\text{def}}{=} w.S' + r.S''$ where $S' \stackrel{\text{def}}{=} r.S'$ and
693 $S'' \stackrel{\text{def}}{=} r.S'' + e.S''$. Initially, memory cell S is empty and offers a choice between a read action
694 r and a write action w . If the write action is performed, the cell receives a value and changes
695 to state S' . In this filled state S' only read actions are admissible to access the stored value,

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

XX:18 Strong Priority and Determinacy in Timed CCS



■ **Figure 5** Symmetric Choice. The final outcome, in states ③ and ④, is non-determinate because it depends on the order of read and write. The continuation 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.

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

705 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
 706 with a single write, both terminating in the empty behaviour 0 . Then, according to the
 707 operational semantics of CCS, we have the two execution sequences $R \mid S \mid W \xrightarrow{\tau} 0 \mid S'' \mid W$
 708 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 $0 \mid S' \mid 0$
 709 are distinct normal forms, in the sense that no reductions are possible anymore. This is pictorially
 710 illustrated in Fig. 5. Observe that the behaviour is not only *non-deterministic* in the sense
 711 that there is more than one scheduling sequence. It is also non-deterministic as the final
 712 outcome depends on the scheduling decisions.

713 Among many CCS extension with priorities, we looked the one of Camilleri and Winskell [7],
 714 referred to as CCS^{CW} in the sequel. CCS^{CW} provide a *prioritised choice* operator $Q \stackrel{def}{=} \alpha_1.Q_1 \triangleright \alpha_2.Q_2$
 715 called “prisum” that gives the action α_1 priority over α_2 and blocks the
 716 second prefix α_2 in case $\alpha_1 = \alpha_2$. Such priorities help us resolve the issue. The problem
 717 with non-determinism originates from the preemptive behaviour of the write-read choice
 718 $w.S' + r.S''$. It permits the read and the write actions to compete with each other for the
 719 next state of the process S . In data-flow and functional programming, we resolve the conflict
 720 by making the write action take precedence over the read one. First, the data-flow variable
 721 must be written by the *producer* process, then it can be read by the *consumers*, if any.
 722 This “write-before-read scheduling policy” is a *prioritised choice*, represented in prioritised
 723 CCS^{CW} [7] as $S \stackrel{def}{=} w.S' \triangleright r.S''$ in which the prisum \triangleright operator enforces a precedence from
 724 left to right. The store S is willing to engage in a read action r (to the right of \triangleright) only if the
 725 environment does not offer a write w (on the left of \triangleright) at the same time, concurrently.

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

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

735 The standard way to implement this scheduling in prioritised CCS is via a transition
 736 relation that keeps track of blocking information. In line with the approaches of [7, 27, 30],
 737 let us write $P \xrightarrow{\alpha}_H P'$ to denote that process P can engage in an action α to continue as P' ,
 738 provided the concurrent environment does not offer any action from H . We will call H the
 739 *blocking set* of the transition. In Plotkin's SOS, the rules for prefix and choice in CCS^{CW} are
 740 the following³:

$$741 \quad \frac{}{\alpha.P \xrightarrow{\alpha}_{\{\}} P} (\text{Act}) \quad \frac{P \xrightarrow{\alpha}_H P'}{P \triangleright Q \xrightarrow{\alpha}_L P'} (\text{PriSum}_1) \quad \frac{Q \xrightarrow{\alpha}_H Q' \quad \tau, \alpha \notin \text{iA}(P)}{P \triangleright Q \xrightarrow{\alpha}_{H \cup \text{iA}(P)} Q'} (\text{PriSum}_2)$$

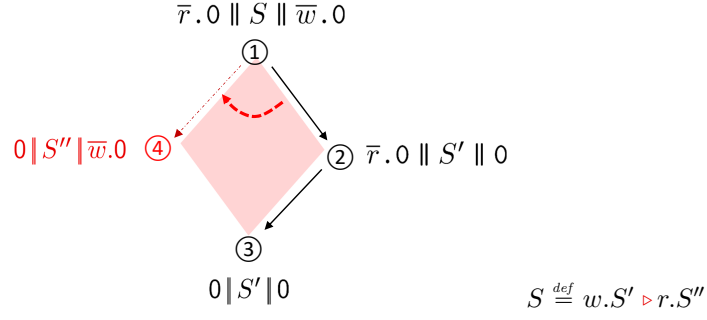
742 where $\text{iA}(P)$ (defined formally later) represents the set of *initial actions* of P that are
 743 offered by P in some environment and \triangleright is the priority sum defined by CCS^{CW} . From
 744 rule (Act) , individual prefixes generate transitions without any blocking constraints.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

$$754 \quad \frac{P_1 \xrightarrow{\alpha_1}_{H_1} P'_1 \quad H_1 \cap \bar{\text{iA}}(P_2) = \{\}}{P_1 | P_2 \xrightarrow{\alpha_1}_H P'_1 | P_2} (\text{Par}_1) \quad \frac{P_2 \xrightarrow{\alpha_2}_{H_2} P'_2 \quad H_2 \cap \bar{\text{iA}}(P_1) = \{\}}{P_1 | P_2 \xrightarrow{\alpha_2}_{H_2} P_1 | P'_2} (\text{Par}_2)$$

$$755 \quad \frac{P_1 \xrightarrow{\alpha_1}_{H_1} P'_1 \quad P_2 \xrightarrow{\alpha_2}_{H_2} P'_2 \quad \alpha_1 = \bar{\alpha}_2 \quad H_1 \cap \bar{\text{iA}}(P_2) = \{\} \quad H_2 \cap \bar{\text{iA}}(P_1) = \{\}}{P_1 | P_2 \xrightarrow{\tau}_{H_1 \cup H_2} P'_1 | P'_2} (\text{Par}_3)$$

757 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
 758 sub-processes P_i . Such is enabled if the competing concurrent process P_{3-i} has no initial
 759 action that is in the blocking set H_i of action α_i , i.e. when the condition $H_i \cap \bar{\text{iA}}(P_{3-i}) = \{\}$
 760 is satisfied. Rule (Par_3) implements the synchronisation between an action α_1 in P_1 and the
 761 associated matching action $\alpha_2 = \bar{\alpha}_1$, originating from P_2 . The side-condition is the same:
 762 each process P_i only accepts the matching action call from P_{3-i} if it does not offer any
 763 action that blocks action α_i . Let us see how this blocking semantics works for our running
 764 example of a write-once memory cell, where we now enforce a write-before-read priority on

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

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

$$767 \frac{\frac{w.S' \xrightarrow{\{w\}} S'}{(Act)}}{S \xrightarrow{\{w\}} S'} (PriSum_1) \quad \text{or} \quad \frac{\frac{r.S'' \xrightarrow{\{r\}} S''}{(Act)} \quad \tau, r \notin iA(w.S')}{S \xrightarrow{\{w\}} S''} (PriSum_2)$$

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

$$770 \frac{\frac{R \xrightarrow{\{r\}} 0}{(Act)} \quad \frac{\vdots}{S \xrightarrow{\{w\}} S''} \quad \{\} \cap i\bar{A}(S) = \{\} \quad \{w\} \cap i\bar{A}(R) = \{\}}{R | S \xrightarrow{\{w\}} 0 | S''} (Par_3)$$

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

$$778 \frac{\frac{R \xrightarrow{\{r\}} 0}{(Act)} \quad \frac{\frac{r.S'' \xrightarrow{\{r\}} S''}{(Act)} \quad S \xrightarrow{\{w\}} S''}{(PriSum_2)}}{R | S \xrightarrow{\{w\}} 0 | S''} (Par_3) \quad \frac{\{w\} \cap iA(W) \neq \{\}}{R | S | W \not\xrightarrow{\{w\}} 0 | S'' | W} (Par_1)$$

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

782 trees in the SOS are as follows

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

783

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

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

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

794 What we want is that every process using only strong priority satisfies the following
795 *weak local confluence* property [18]. Suppose P admits two transitions to different states
796 $P_1 \neq P_2$ 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 $\alpha_1 \notin H_2$
797 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
798 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
799 to express non-interference between transitions. Then, our weak⁴ local confluence expresses
800 a weak “diamond property” stating that every diverging choice of non-interfering transitions
801 immediately reconverges. The subset inclusions $H'_i \subseteq H_i$ ensure that delaying an admissible
802 action does not introduce more chances of interference.

803 A.2.1 Fix 1

804 The standard notion of priority for CCS, as implemented in CCS^{CW} or CCS^{Ph} , of course was
805 not designed to restrict CCS for determinism but to extend it conservatively to be able to
806 express scheduling control. Even in the fragment of CCS^{CW} where all choices are prioritised,
807 weak confluence fails. For instance, although our memory cell S behaves deterministically
808 for a single reader R and a single writer W , in the context $E \stackrel{def}{=} W_1 | [\cdot] | W_2$ of two writers
809 $W_i \stackrel{def}{=} \bar{w}.W'_i$, it creates a race. The two writers compete for the single write action on S
810 and only one can win. Using the operational semantics of CCS^{CW} we can derive the (non-
811 interfering) transitions $W_1 | S | W_2 \xrightarrow{\{ \}} W'_1 | S' | W_2$ and $W_1 | S | W_2 \xrightarrow{\{ \}} W_1 | S' | W'_2$. Still,
812 the two outcomes of the divergence are a critical pair that have no reason to reconverge
813 $W'_1 | S' | W_2 \rightarrow P'$ and $W_1 | S' | W'_2 \rightarrow P'$ if W'_i are arbitrary behaviours. To preserve confluence
814 we must block the environment from consuming the single prefix $w.S'$ twice. The problem is
815 that the synchronisations $W_1 | S \xrightarrow{\{ \}}_{H_1} W'_1 | S'$ and $S | W_2 \xrightarrow{\{ \}}_{H_2} S | W'_2$ have empty blocking

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

XX:22 Strong Priority and Determinacy in Timed CCS

816 sets $H_i = \{\}$ which does not prevent the second writer W_{3-i} to be added in parallel. If
 817 $w \in H_i$, then the rules (Par_{3-i}) would automatically prevent the second reader. Now, suppose
 818 we extend the action rule (Act) to expose the singleton nature of a prefix, like by (Act^+),
 819 $\alpha.P \xrightarrow{\{\alpha\}} P$ we would not only prevent a second concurrent writer but already a single writer.
 820 The transitions $W_1 \xrightarrow{\{\bar{w}\}} W'_1$ and $S \xrightarrow{\{w\}} S'$ would not be able to communicate, violating
 821 both side-condition $\{\bar{w}\} \cap \bar{iA}(S) = \{\}$ and $\{w\} \cap \bar{iA}(W_1) = \{\}$ of (Par_3). The issue is that
 822 in the setting of CCS^{CW} (and CCS^{Ph}) the blocking set H of a transition $P \xrightarrow{\alpha_H} P'$ expresses
 823 a priority rather than an interference. An action cannot take priority over itself without
 824 blocking itself. Therefore, in CCS^{CW} (and CCS^{Ph}) we always have $\alpha \notin H$. Here, we need to
 825 apply a weaker interpretation. The actions in H only prevent α from synchronising with
 826 a matching $\bar{\alpha}$ in the environment, if they are executed in a *different* (and thus competing)
 827 prefix from the one that executes $\bar{\alpha}$. Actions in H *interfere* with α in the sense that they are
 828 initial actions of P sharing the same control thread in which α is offered. Thus, we want to
 829 avoid a situation in which the environment synchronises with $\bar{\alpha}$ in the presence of *another*
 830 *distinct* communication on some $\beta \in H$. For then we would have a race for the behaviour of
 831 the thread shared by α and β in P .

832 To fix this, we need to refine the side-conditions $H_i \cap \bar{iA}(P_{3-i})$ in (Par_3) to check
 833 not *all* initial actions $\bar{iA}(P_{3-i})$ but only those initial actions of P_{3-i} that are *competing*
 834 with the matching transitions $P_1 \xrightarrow{\alpha_1} P'_1$ and $P_2 \xrightarrow{\alpha_2} P'_2$ with $\alpha_1 = \bar{\alpha}_2$, involved in the
 835 communication $P_1 | P_2 \xrightarrow{\tau} P'_1 | P'_2$ that we want to protect. We fix this by refining the
 836 information exposed by a transition. More specifically, we extend each transition $P \xrightarrow{\alpha_H} P'$
 837 by the set K of initial actions that are offered in competing prefixes of the same thread as α
 838 or by threads concurrent to the thread from which α is taken. Then, the set of all initial
 839 actions is simply $iA(P) = K \cup \{\alpha\}$. Let us call K the *initial environment* of the transition.
 840 From the description, we expect the invariant that $H \subseteq K$.

841 ▶ Remark 25. Note that we cannot simply define $K \stackrel{def}{=} iA(P) - \{\alpha\}$, because α could be
 842 contained in K . For example, take $P \stackrel{def}{=} \alpha.0 | \alpha.0$, where $P \xrightarrow{\{\alpha\}} \alpha.0$. The blocking set is
 843 $H = \{\alpha\}$ since the prefix is volatile and must be protected from a competing consumptions.
 844 Also, we need α to be in the initial environment $K = \{\alpha\}$, because the transition's action α
 845 is offered a second time in a concurrent thread. This is different from the set $iA(P) - \{\alpha\} =$
 846 $\{\alpha\} - \{\alpha\} = \{\}$. This shows that we must compute the initial environment explicitly as an
 847 extra label with each transition.

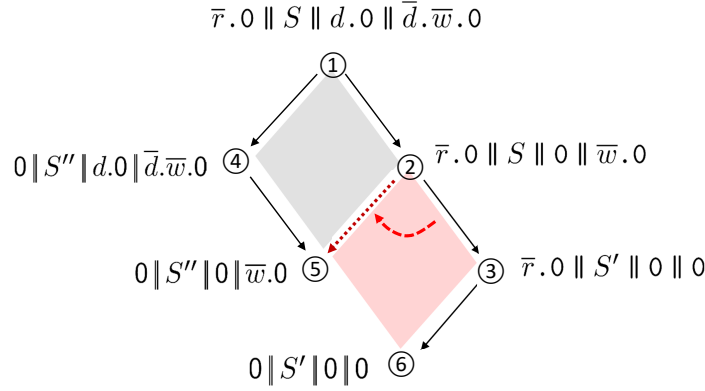
848 Now we can reformulate the parallel rule as

$$849 \frac{P_i \xrightarrow[\bar{K}_i]{\alpha_i} P'_i \quad \alpha_1 = \bar{\alpha}_2 \quad H_i \cap \bar{K}_{3-i} = \{\} \quad \text{for } i \in \{1, 2\}}{P_1 | P_2 \xrightarrow[\bar{K}_1 \cup \bar{K}_2]{\tau} P'_1 | P'_2} (Par_3^+)$$

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

852 A.2.2 Fix 2

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



■ **Figure 7** The store S in composition with a reader $\bar{r}.0$ 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.

858 we obtain a determinate process algebra?” Indeed, we can show that in the “asynchronous”
 859 fragment of CCS, i.e. where all prefixes $\alpha.P$ are of form $\alpha.0$, confluence can be granted: On
 860 the contrary, in the “synchronous” fragment of CCS, where actions sequentially guard new
 861 actions, we run into a problem of non-monotonicity. A communication between prefixes
 862 may trigger new actions that block another communication that was enabled before. In
 863 other words, it is not guaranteed that independent actions remain admissible. As an
 864 example, in the above reader-writer scenario, the actions a_i ($i \in \{1, 2\}$) may become
 865 blocked in processes P_{3-i} . Now, consider a parallel composition $R \mid S^* \mid W^*$ in which a
 866 modified store $S^* \stackrel{\text{def}}{=} S \mid d.0$ offers another concurrent method action d on the side in
 867 parallel. Let the writer W^* be delayed by a synchronisation with this concurrent action
 868 d , i.e., $W^* \stackrel{\text{def}}{=} \bar{d}.W \stackrel{\text{def}}{=} \bar{d}. \bar{w}.0$. The scheduling sequences are depicted in Fig. 7. We can
 869 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
 870 $R \mid S \mid d.0 \xrightarrow[\{w, d\}]{\tau}_{\{\bar{r}, r, w\}} S'' \mid d.0$ can now take place by (Par_3) before the write to give the
 871 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
 872 first reduction is justified by rule (Par_1), because the side-condition $\{w\} \cap \bar{i}\bar{A}(W) = \{w\} \cap \{d\} =$
 873 $\{\}$ is satisfied. The blocking side-condition which only looks at the moment when the read is
 874 happening does not see the write action \bar{w} by W^* hidden behind its initial action \bar{d} . It lets
 875 process R access and read the store S . As before, however, the following reduction sequence is
 876 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
 877 write action happens before the read action. This creates two diverging reduction sequences
 878 (incoherent schedules) in which the final store is S' or S'' , non-deterministically.

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

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

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

$$902 \frac{P \xrightarrow[E]{a}_{H \cup L} P' \quad E' = E | Q \quad H \cap \overline{\text{iA}^\tau}(E') = \{\}}{P | Q \xrightarrow[E']{a}_{H \cup L} P' | Q} (Par_1^*)$$

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

$$905 \frac{P \xrightarrow[E_1]{a}_{H_1 \cup L_1} P' \quad Q \xrightarrow[E_2]{\bar{a}}_{H_2 \cup L_2} Q' \quad E = E_1 | E_2 \quad \forall i. H_i \cap (\overline{\text{iA}^\tau}(E) \cup L_{3-i}) = \{\}}{P | Q \xrightarrow[E]{\tau}_{H_1 \cup H_2 \cup L_1 \cup L_2} P' | Q'} (Par_3^*)$$

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

$$907 R \xrightarrow{\tau}_0 \{r\} \cup \{ \} R' \text{ and } S | d.0 \xrightarrow{\tau}_{d.0} \{w,r\} \cup \{ \} S'', \text{ we get } R | S | d.0 \xrightarrow{\tau}_{d.0} \{w,r\} \cup \{ \} S'' | d.0.$$

908 The side-condition of (Par_3^*) is satisfied since the “weak” initial action set (where τ actions are
 909 dropped) $\text{iA}^\tau(d.0) = \{d\}$. Thus, both $\{r\} \cap \overline{\text{iA}^*}(d.0) = \{\}$ as well as $\{w,r\} \cap \overline{\text{iA}^*}(d.0) = \{\}$.
 910 If now want to shunt this past the writer process W^* using rule (Par_1^*), then we get
 911 stuck. Since $d.0 | W^* \xrightarrow{\tau}_0 \{d,\bar{d}\} \cup \{ \} W$ and $\bar{w} \in \text{iA}(W)$, we find $\bar{w} \in \text{iA}^\tau(d.0 | W^*)$ and thus
 912 $\{w\} \cap \overline{\text{iA}^*}(d.0 | W^*) \neq \{\}$. Therefore, the side-condition of (Par_1^*) is violated and the
 913 reading interaction of $R | S | d.0$ cannot be composed with W^* . This is what we want.
 914 The transition from state ② to state ③ in Fig. 7 is now blocked. On the other hand,
 915 the reduction $R | S | d.0 \xrightarrow[d]{}_{R | S} \{d\} \cup \{ \} R | S$ can be composed with $W^* \xrightarrow{\bar{d}}_0 \{\bar{d}\} \cup \{ \} W$ via (Par_3^*)
 916 to make a reduction $R | S | d.0 | W^* \xrightarrow{\tau}_{R | S} \{d,\bar{d}\} \cup \{ \} R | S | W$. Regarding the side-condition, we
 917 notice that the weak initial action set $\text{iA}^\tau(R | S)$ of the reduction context is irrelevant, because
 918 it does not contain d or \bar{d} . In configuration $R | S | W$ (state ② in Fig. 7), the reader R cannot
 919 interact with S . The associated reduction $R | S | W \xrightarrow{\tau}_W \{r,w\} \cup \{ \} S'' | W$ is blocked by the
 920 reduction environment, because $w \in \overline{\text{iA}^*}(\bar{w}.W')$. For our further discussion, let us call the
 921 proposed change in the side-conditions leading to (Par_i^*) from (Par_i) a process algebra with
 922 *strong priorities*⁵ and let's call the traditional process algebra with priorities, introduced by

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

923 [7, 30], as process algebra with *weak priorities*.

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

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

933 Obviously, to compute $iA^\tau(E)$ we need to know the full transition relation for E . In
 934 other words, the existence of a transition of a process P depends on the absence of certain
 935 transitions of a certain part of process P . If P is defined recursively or via repetition then
 936 the behaviour of a part of P might depend on the behaviour of P itself. Does such a
 937 self-referential semantical definition make sense at all? How do we ensure the absence of a
 938 Russell-paradox style relation, by defining a process $\text{rec } p.P$ to have a transition to P' if it
 939 does not have a transition to P' for a suitable P' ? Such would be plainly inconsistent at
 940 the meta-level, because it would not only define simply an empty transition relation⁶, but it
 941 would not define a unique semantic relation at all.

942 In the standard theory of weakly prioritised process algebra as described in the cited
 943 references of CCS with priorities, the problem does not exist, because the set of (strong)
 944 initial actions $iA(E)$ can be determined by structural recursion on E without any reference
 945 to the transition relation. Unfortunately, for strong prioritized process algebra as defined in
 946 this paper, this is not possible anymore. So, how do we make sense of transition-generating
 947 rules (Par_i^*) that depend on negative premises? One option is to treat the negation in the
 948 side-condition $H \cap \overline{iA^*}(E) = \{\}$ constructively rather than classically, say like negation in
 949 normal logic programming. This would amount to building a second independent (inductive)
 950 rule system to generate constructive judgements about the absence of weakly reachable
 951 transitions. Trouble is that there is no canonical semantics of constructive negation, even in
 952 logic programming⁷.

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

958 A more perspicuous attack is to keep the classical set-theoretic interpretation but use an
 959 over-approximation $L \cap \overline{\text{fiA}^*}(E) = \{\}$ where $\text{fiA}^*(E) \supseteq iA^\tau(E)$ is intuitively a superset of
 960 weakly reachable initial sender actions obtained from a relaxed transition relation, formally
 961 defined later in this paper. A natural over-approximation that can be defined independently
 962 is the “free speculative” scheduling that ignores all priorities. The stronger side-condition
 963 $L \cap \overline{\text{fiA}^*}(E) = \{\}$ forces the absence of a blocking sender action under the worst-case
 964 assumption of priority-free scheduling. Obviously, this is sound but will have the effect that
 965 some programs block that could make progress under a tighter approximation of the sets

⁶ 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.

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

966 $iA^\tau(E)$. Between $fiA^*(E)$ and $iA^\tau(E)$ there be a whole range of different semantics that may
 967 or may not be equivalent to specific constructive semantics of negation. Exploring the range
 968 of options systematically will certainly be a worthwhile research exercise likely to cover much
 969 new ground in the theory of process algebras with priorities.

970 A.3 Synchronous CCS with Strong Priorities and Clocks is Confluent

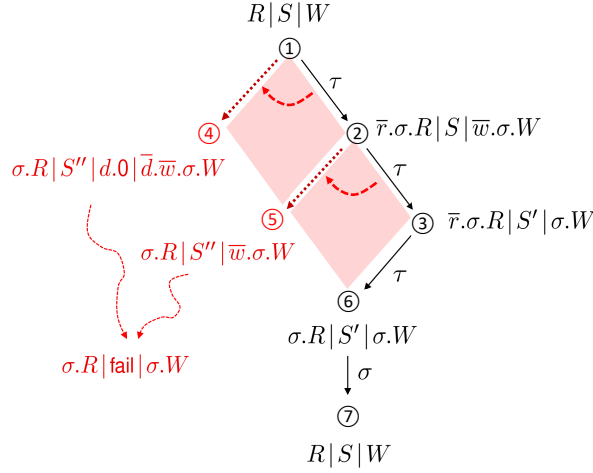
971 We argue that the use of priorities for determinacy calls for a combination of strong priorities
 972 with a second important concept that has been investigated in the literature before, but
 973 independently. This is the well-known notion of a *clock*, from Timed Process Algebras, like
 974 e.g.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

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

986 To get this off the ground we introduce a clock broadcast action σ to separate each round
 987 of communication. We define the recursive processes $W \stackrel{def}{=} d.0 \mid \bar{d}.\bar{w}.\sigma.W$ and $R \stackrel{def}{=} \bar{r}.\sigma.R$
 988 for one write and one read per macro-step. Assuming the content of the memory variable
 989 is reset for every tick, we can define S as follows $S \stackrel{def}{=} w.S' + r:\{w\}.S'' + \sigma:\{w, r\}.S$ with
 990 $S' \stackrel{def}{=} r.S'' + \sigma:\{r\}.S'$ and $S'' \stackrel{def}{=} w.fail + r:\{w\}.S'' + \sigma:\{w, r\}.S$. The clock always takes *lowest*
 991 priority among all other rendez-vous actions out of a state. The idea is that the clock should
 992 fire only if the system has stabilised and no other admissible data actions are possible. This
 993 is called *maximal progress* and it is a standard assumption in timed process algebras [15, 8].
 994 The fairly standard rules for clock actions, inspired by timed process algebras are then:

$$\begin{array}{c}
 \frac{}{\sigma.P \xrightarrow{\sigma} \{ \} P} (Clk_1) \quad \frac{P \xrightarrow{\sigma}_{R_1} L_1 P' \quad P \xrightarrow{\sigma}_{R_2} L_2 P' \quad (L_1 \cup L_2) \cap iA^\tau(R_1 \mid R_2) = \{ \}}{P \mid Q \xrightarrow{\sigma}_{R_1 \mid R_2} L_1 \cup L_2 P' \mid Q'} (Clk_2)
 \end{array}$$

996 Those two rules enforces global synchronisation and make the clock deterministic. As such,
 997 the evaluation of $R \mid S \mid W$ “prunes” all reductions that could lead to non confluent reductions.
 998 Fig. 8 shows how higher priorities can introduce determinism in scheduling. The importance
 999 of the clock σ is that its associated synchronisation rule (Clk_2) enforces, in a quite natural way,
 1000 the barrier explicitly in the SOS rather than making it an artefact of a specific synchronisation
 1001 process, as such obliging the programmer to an extra effort. The fact that the broadcasting
 1002 clock is hard-wired into the rules can now be exploited for the computation of the *weak*
 1003 *initial action sets*, defined formally $iA^\tau(R)$. More precisely, we only need to predict the
 1004 behaviour of R for the current macro-step, i.e., up to the next synchronization of σ by R .
 1005 For a transition $P \xrightarrow{a}_R L P'$ we know that all actions taking place after σ in R are definitely
 1006 observed to happen in the next macro step and thus not in competition with the current
 1007 action a under blocking set L . In this fashion, the clock acts as as a *stopper sentinel* for
 1008 prediction and evaluation of the blocking set L relative to the reduction environment R . If
 1009 every recursion is instantaneous, then the search space becomes finite.



■ **Figure 8** Strong Priorities introduce confluence in scheduling.

1010 As a final remark, for the purposes of this paper, adding synchrony and determinism in
 1011 process algebras, we are not (and indeed we do not need to be) interested in higher-order
 1012 calculi, like e.g. the π -calculus [24, 25], where processes are *first-class* objects, i.e. sent
 1013 and received as a values, as in $a!P.Q$ and $a?x.(x|Q)$. In the next sections, we present the
 1014 syntax, the structural operational semantics à la Plotkin, we set up all the definitions and
 1015 the metatheory necessary to formally prove the claimed informal properties.

1016 B Examples, new and reloaded

1017 This section contains a number of reloaded and new examples in view of the definitions and
 1018 results presented in the previous section.

1019 The weakly-enabled transitions of unlocked, free processes coincide with the semantics
 1020 of genuine Milner's CCS [22], as stated in the following easy exercise.

1021 ▶ **Exercise 26** (On Def. 3). *If P is unlocked and free, then a transition $P \xrightarrow{\alpha}_R Q$ is weakly
 1022 enabled iff $P \xrightarrow{\alpha} Q$ is admissible, i.e., iff it is derivable in the original Milner's SOS for CCS.*

1023 In fact, one can show that the semantics of weak enabling generates the theory of
 1024 CCS^{Ph} [27]. To be more precise, let us call a process P *irreflexive* if no action prefix $\alpha:H.Q$
 1025 occurring in P has $\alpha \in H$, i.e., blocks itself. In the semantics of CCS^{Ph} , such *self-blocking*
 1026 prefixes do not contribute any transitions. Thus, CCS^{Ph} captures a theory of irreflexive
 1027 processes. We leave as a less easy exercise that all weakly enabled transitions between
 1028 irreflexive processes are precisely the ones allowed in CCS^{Ph} .

1029 ▶ **Exercise 27** (On Def. 3). *If P is unlocked and irreflexive, then a transition $P \xrightarrow{\alpha}_R Q$ is
 1030 weakly enabled iff $P \xrightarrow{\alpha}_H P'$ is derivable in the operational semantics of CCS^{Ph} .*

1031 Thus, from the point of view of CCS^{spt} , theories of CCS and CCS^{Ph} correspond to theories
 1032 of free and irreflexive processes, respectively, under weak enabling. Here we suggest two
 1033 extensions to this classical work. Firstly, we apply a stronger scheduling restriction, called
 1034 *strong enabling* to achieve confluence and secondly we make use of self-loops to control
 1035 sharing.

XX:28 Strong Priority and Determinacy in Timed CCS

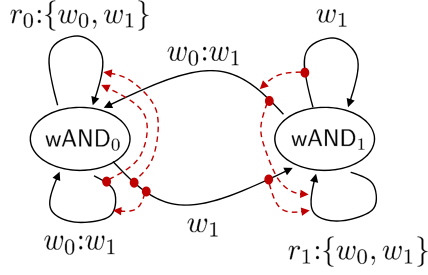
1036 ▶ **Example 28** (On Def. 16). If $A \not\equiv B$ then $a.A + a.B$ is not coherent while $a:a.A | a:a.B$
 1037 is coherent. Conflicting choices on the same label in a single thread cannot be externally
 1038 observed and thus cannot be deterministically scheduled via their externally observable
 1039 properties. In coherent processes such choices are assumed to be resolved internally by each
 1040 thread itself, rather than through the precedence relation. Note that the process $a:a.A + a:a.B$
 1041 in which the two action prefixes are self-blocking, is not coherent. The reason is that the
 1042 two identical (self-blocking) transitions lead to structurally distinct states $A \not\equiv B$. Thus, the
 1043 confluence condition expressed in Def. 16 applies and A and B should be reconvergent with
 1044 transitions $A \xrightarrow{a} Q'$ and $B \xrightarrow{a} Q'$. But this is not guaranteed for arbitrary A and B . On the
 1045 other hand, a single self-blocking prefix $a:a.A$ is coherent. Without the self-blocking, $a.A$ is
 1046 only coherent if process A permits infinite repetition of a as in $A \stackrel{def}{=} a.A$ or $A \stackrel{def}{=} a | A$.

1047 The precedence relation cannot resolve the choice of a prefix with itself. The precedence
 1048 relation can resolve the choice between *distinct* actions prefixes. For instance, if $a \neq b$ then
 1049 $a.A + b.B$ is not coherent, for any A and B . However, both $a:a.A | b:b.B$ and $a:a.A + b:\{a, b\}.B$
 1050 are coherent. In the former case, the actions are concurrently independent and so confluence
 1051 occurs naturally. In the latter case, the choice is resolved by precedence: No reconvergence
 1052 is required by coherence Def. 16 as the actions are interfering with each other. ◀

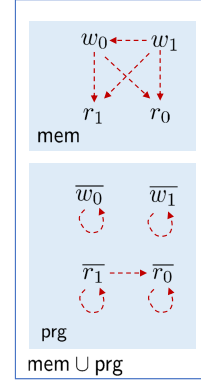
1053 ▶ **Example 29** (On coherence, pivotable and policies). The process $P_1 \stackrel{def}{=} a | \bar{b} | \bar{a}:b + b$ is
 1054 coherent and pivotable with a policy π such that $\pi \Vdash b \dashrightarrow \bar{a}$ and $\pi \Vdash \bar{b} \diamond a$. The rendez-vous
 1055 synchronisation $\tau = a | \bar{a}$ is blocked by the presence of the offering \bar{b} from a *third* thread. We
 1056 have $P_1 \xrightarrow{\tau}_{\bar{b} \{b\}} \bar{b}$, which is not weakly enabled, because $\{b\} \cap i\bar{A}(\bar{b}) \neq \{\}$. The blocking is due to
 1057 the reaction environment \bar{b} of the transition. For contrast, the process $P_2 \stackrel{def}{=} (a + \bar{b}) | (\bar{a}:b + b)$
 1058 is pivotable but not coherent. In our semantics, the synchronisation of a and $\bar{a}:b$ in P_2
 1059 is blocked by the side-condition of rule (Par_3) which introduces τ into the blocking set
 1060 $P_2 \xrightarrow{\tau}_{\bar{0} \{b, \tau\}} 0$ because $\{b\} \cap i\bar{A}(a + \bar{b}) \not\subseteq \{\bar{a}\}$. This transition is not weakly enabled, because
 1061 of $\{b, \tau\} \cap (i\bar{A}(0) \cup \{\tau\}) \neq \{\}$. We can make the pivotable P_1 also coherent by forcing a
 1062 sequential ordering between a and \bar{b} in the first thread, say $P_3 \stackrel{def}{=} a.\bar{b} | (\bar{a}:b + b)$, then the
 1063 synchronisation is even strongly enabled $P_3 \xrightarrow{\tau}_{\bar{0} \{b\}} \bar{b}$. If we introduce a precedence into the
 1064 first thread, say $P_4 \stackrel{def}{=} (a:b + \bar{b}) | (\bar{a}:b + b)$, then we are also coherent but no longer pivotable.
 1065 Like for P_2 the synchronisation $\tau = a | \bar{a}$ is blocked and not weakly enabled, because of the
 1066 introduction of τ into the blocking set by the side condition of Par_3 . ◀

1067 ▶ **Example 30** (On Normal form). For the process $P_1 \stackrel{def}{=} (a.c | b.d | \bar{a} | \bar{b}) \setminus \{a, b\}$ we have
 1068 $(a.c | d | \bar{a}) \setminus \{a, b\} \xleftarrow{\tau} P_1 \xrightarrow{\tau} (c | b.d | \bar{b}) \setminus \{a, b\}$ where the continuation processes $(c | b.d | \bar{b}) \setminus \{a, b\}$
 1069 and $(a.c | d | \bar{a}) \setminus \{a, b\}$ are not structurally equivalent. They are not even behaviourally
 1070 equivalent. However, both reductions are concurrently independent and commute with each
 1071 other. We can execute both rendez-vous synchronisations $\tau = a | \bar{a}$ and $\tau = b | \bar{b}$ in any order,
 1072 the process P converges to a unique final outcome $(c | d) \setminus \{a, b\}$. Similarly, in the multi-cast
 1073 process $P_2 \stackrel{def}{=} (a.c | a.d | \bar{a} | \bar{a}) \setminus \{a\}$ the reduction behaviour is not structurally deterministic
 1074 but nevertheless generates a unique normal form. ◀

1075 ▶ **Example 31** (On Clock-hiding). The process $P \stackrel{def}{=} \sigma + a:\sigma | \sigma$ offers a choice of two transitions
 1076 and $P \xrightarrow{a}_{\sigma \{ \sigma \}} \sigma$. These transitions interfere with each other as the σ -transition blocks the
 1077 a -transition. We expect the a -transition to be blocked. This must happen when the clock
 1078 is hidden and thus generates a silent step. Indeed, when we hide the clock, we change the
 1079 blocking σ into a blocking $\sigma / \sigma = \tau$ and obtain $P / \sigma \xrightarrow{a}_{\sigma / \sigma \{ \tau \}} \sigma / \sigma$. Now the τ in the blocking
 1080 set eliminates the a -transition under enabling semantics, in particular under constructive



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



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

■ **Figure 9** A boolean memory process.

1081 scheduling. So far so good. However, we can replace the blocking set $\{\sigma\}$ by $\{\tau\}$ only if the
 1082 clock is actually executable in the concurrent environment $0 \mid \sigma$. But it is not. In the process
 1083 $Q \stackrel{def}{=} (\sigma + a:\sigma \mid b.\sigma) / \sigma$ the local clock σ also blocks the action a by priority. For the clock
 1084 to become enabled, however, it needs the synchronisation with an external action \bar{b} . The
 1085 naive hiding rule would generate the transition $Q \xrightarrow{(b.\sigma)/\sigma}_{\{\sigma\}} (b.\sigma) / \sigma$. Under constructive
 1086 scheduling we would find $\sigma \in \bar{iA}^*(b.\sigma)$. This could be taken as justification to introduce τ
 1087 into the blocking set. It provides enough information for us to be able to see that the parallel
 1088 composition $Q \mid \bar{b}$ will not permit the a -transition any more.

1089 ▶ **Example 32** (On Pivable Processes). The process $a + b:c \mid \bar{a} + \bar{b}:\bar{a}$ is pivable. If P
 1090 conforms to π then the only precedences forced are $\pi \Vdash c \dashrightarrow b$ and $\pi \Vdash \bar{a} \dashrightarrow \bar{b}$ while it
 1091 satisfies $\pi \Vdash \bar{c} \diamond \bar{b}$ and $\pi \Vdash a \diamond b$. However, the process is not input-scheduled, because it has
 1092 $\pi \Vdash \bar{a} \dashrightarrow \bar{b}$ while $\pi_{is} \Vdash \bar{a} \diamond \bar{b}$, assuming $a \neq b$.

1093 ▶ **Example 33** (On Coherent and Pivable Processes). The process $P_1 \stackrel{def}{=} \sigma + a:\sigma \mid \bar{a}$ is
 1094 coherent and pivable, with policy $\pi \Vdash \sigma \dashrightarrow a$ and $\pi \Vdash \sigma \diamond \bar{a}$. It has both $a, \bar{a} \in iA(P)$
 1095 but $\sigma \notin iA(P)$ since the clock is not offered by the parallel process \bar{a} . The synchronisation
 1096 $\tau = a \mid \bar{a}$ 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
 1097 actions $\sigma, a, \bar{a} \in iA(P_2)$ initial. It is coherent but not pivable. Since both a and \bar{a} are in a
 1098 preemption relationship with σ , P_2 does not satisfy concurrent independence of σ with neither
 1099 a nor \bar{a} . Formally, any policy for P_2 must have $\pi \Vdash \sigma \dashrightarrow a$ and $\pi \not\Vdash \bar{a} \dashrightarrow \sigma$ violating the
 1100 pivot requirement. The same is true of the process $P_3 \stackrel{def}{=} \sigma + (a:\sigma \mid \bar{a})$ which can perform
 1101 both σ and a (silent) reduction that has σ in its blocking set.

1102 ▶ **Example 34** (On Read/Write Process, reloaded). Figure 9 show that the processes $wAND_v$
 1103 conform to a policy mem with $mem \Vdash w_i \dashrightarrow r_j$ for $i \neq j$ and $mem \Vdash w_1 \dashrightarrow w_0$. Obviously,
 1104 mem is a pivot policy. The processes $wAND_v$ are also coherent. One verifies by simple case
 1105 analysis that for any derivative Q of $wAND_v$, if $Q \xrightarrow{\ell}_H Q'$ and $\ell \notin H$ then there is a repetition
 1106 $Q' \xrightarrow{\ell}_{H'} Q''$ with $H' = H$. This yields Def. 16(1). Secondly, suppose $Q \xrightarrow{\ell_1}_H Q_1$ and $Q \xrightarrow{\ell_2}_H Q_2$.
 1107 Then, obviously, if $\ell_1 \neq \ell_2$ the two transitions are never interference-free, because of the
 1108 choice of blocking sets which orders all summation prefixes in a linear precedence order. If
 1109 $Q_1 \neq Q_2$ the necessarily $\ell_1 \neq \ell_2$. Thus, Def. 16(2) is satisfied, too, for trivial reasons. So, the
 1110 memory $wAND_v : mem$ is pivable. Note that $wAND_v$ is not a discrete process, since none of

XX:30 Strong Priority and Determinacy in Timed CCS

1111 its actions is self-blocking. We can put the memory in parallel with arbitrarily many writers
1112 and readers. Consider a parallel composition $\mathbf{wAND}_v | P$ where P is any single-threaded
1113 process that synchronises via \bar{w}_i for writing and \bar{r}_i for reading. Each writing action of P is a
1114 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
1115 continuation processes R_0 and R_1 are selected according to the content of the memory, by
1116 matching with the r_0 action of \mathbf{wAND}_0 or the r_1 action of \mathbf{wAND}_1 . If the memory would offer
1117 both a synchronisation on r_0 and r_1 (which it does not), the reader would take the match on
1118 r_1 , because of the precedences. If Q , R_0 and R_1 follow the same principle, such a sequential
1119 program is a discrete process $P : \mathbf{prg}$ that is coherent and conformant to the policy \mathbf{prg} with
1120 $\mathbf{prg} \Vdash \bar{r}_1 \dashrightarrow \bar{r}_i$ and $\mathbf{prg} \Vdash \bar{r}_0 \dashrightarrow \bar{r}_0$. Now we observe that the union $\mathbf{mem} \cup \mathbf{prg}$ of policies
1121 is a pivot policy. Since $\mathbf{wAND}_v : \mathbf{mem}$ and $P : \mathbf{prg}$ we have $\mathbf{wAND}_v : \mathbf{mem} \cup \mathbf{prg}$ as well as
1122 $P : \mathbf{mem} \cup \mathbf{prg}$. Therefore, by Prop. 51 we get $\mathbf{wAND}_v | P : \mathbf{mem} \cup \mathbf{prg}$ i.e., the composition
1123 is coherent and pivotable. In fact we can put the memory in parallel with arbitrarily many
1124 sequential programs coherent for \mathbf{prg} and the composition $\mathbf{wAND}_v | P_1 | P_2 | \dots | P_n$ will be
1125 coherent for $\mathbf{mem} \cup \mathbf{prg}$. In particular, all reductions will be confluent under strong enabling.
1126 Technically, what happens is that first all writers 1, synchronising as $w_1 | \bar{w}_1$, will go ahead.
1127 Then all writers of 0, synchronising as $w_0 | \bar{w}_0$ become enabled. Only then the readers with
1128 be able to retrieve the final value via $r_0 | \bar{r}_0$ or $r_1 | \bar{r}_1$ rendez-vous synchronisations.

1129 Observe that the pivot policy $\mathbf{mem} \cup \mathbf{prg}$ permits a choice (and precedence) between
1130 both the receiver actions w_0, w_1 but also between the sender actions \bar{r}_0, \bar{r}_1 . As a result, the
1131 combined memory and program, which is coherent for $\mathbf{mem} \cup \mathbf{prg}$, is not longer an input-
1132 scheduled process. This breaks the assumption on which the theory of classical prioritised
1133 process algebras like [7] or [30] are based. In reference to the former, Phillips writes:

1134 *They [Camilletti and Winskel] decide to sidestep this question by breaking the symmetry*
1135 *in CCS between inputs and outputs, and only allowing prioritised choice on input*
1136 *actions. We feel that this complicates the syntax and operational semantics, and should*
1137 *not be necessary. [27]*

1138 Our example and analysis indeed highlights the usefulness of the more general setting
1139 suggested by Phillips. In order to explain the theory of deterministic shared memory we
1140 need a larger class of processes than those considered in CCS^{CW} .

1141 The memory cell schedules readers after the writers and among the writers prefers to
1142 engage first with those that set the value to 0, in the prioritised sum. This implements
1143 an Esterel-style resolution function for multi-writer convergecast, multi-reader broadcast of
1144 boolean signals. All concurrent readers receive the same value, which is the combined-AND
1145 of all values written by any concurrent writer. If any process writes a 1 it will win the
1146 competition. This is also known as a “wired-AND”. Likewise, we can implement a memory
1147 cell that models a “wired-OR” if we swap the roles of w_0 and w_1 in the prioritised choices of
1148 \mathbf{wAND}_v .

1149 Note that the wired-AND behaviour only applies to concurrent threads. A single thread
1150 can use the \mathbf{wAND} cell as a local memory and destructively update it sequentially as it
1151 wishes. Consider the difference: Both *concurrent* writers $\bar{w}_1:\bar{w}_1 | \bar{w}_0:\bar{w}_0 | \mathbf{wAND}_0 | R$ and
1152 $\bar{w}_0:\bar{w}_0 | \bar{w}_1:\bar{w}_1 | \mathbf{wAND}_0 | R$ will necessarily make the reader $R \stackrel{\text{def}}{=} r_0:r_1.A + r_1:\{r_0, r_1\}.B$ detect
1153 a value r_0 and therefore end in state A (wired-AND of w_0 and w_1). The two single-threaded
1154 *sequential* writings $\bar{w}_1:\bar{w}_1.\bar{w}_0:\bar{w}_0 | \mathbf{wAND}_0 | R$ and $\bar{w}_0:\bar{w}_0.\bar{w}_1:\bar{w}_1 | \mathbf{wAND}_0 | R$ will generate two
1155 different readings by R : The first is the reading r_0 , leading to A . The second is the reading
1156 r_1 , leading to B .

1157 ▶ **Example 35** (On ABRO, reloaded). One can verify that the process ABRO is coherent and

1158 pivotable.

1159 ■ The choices in each of the threads R, A, B, O are fully resolved by the blocking sets,
 1160 i.e., every pair of actions in the summations are distinct and in precedence relationships.
 1161 Every rendez-vous prefix is self-blocking unless the action can be repeated with the same
 1162 blocking in the continuation process. This happens for channels s and k in the recursions
 1163 $O \stackrel{def}{=} \dots + s:k.O + \dots$ and $R' \stackrel{def}{=} \bar{k}.R' + \dots$. Thus, the threads R, A, B, O themselves are
 1164 coherent.

1165 ■ The associated policy π , for which the threads R, A, B, O are coherent, enforces the
 1166 precedences

- 1167 ■ $\pi \Vdash k \dashrightarrow \ell$ for $\ell \in \{k, a, b, s, t, \sigma\}$
- 1168 ■ $\pi \Vdash \ell \dashrightarrow \ell$ for $\ell \in \{r, k, a, b, \bar{s}, t, \bar{\sigma}\}$
- 1169 ■ $\pi \Vdash s \dashrightarrow t$
- 1170 ■ $\pi \Vdash \ell \dashrightarrow \sigma$ for $\ell \in \{a, b, s, t\}$.

1171 The dual matching pairs for these precedences, however, are all concurrently independent.
 1172 More precisely, we look at distinct labels, $\pi \Vdash \bar{k} \diamond \bar{\ell}$ for all $\ell \in \{a, b, s, t, \sigma\}$, $\pi \Vdash \bar{s} \diamond \bar{t}$ and
 1173 $\pi \Vdash \bar{\ell} \diamond \sigma$ for $\ell \in \{a, b, s, t\}$.

1174 ■ Thus the policy π (and thus $\pi \setminus \mathcal{R}$) is a pivot policy and so the parallel composition
 1175 $A | B | O$ is coherent for π . Further, π is precedence-closed for $\{s, t\}$ which means that
 1176 ABO and also ABRO are coherent for π .

1177 ▶ **Example 36** (On Sequentiality à la Milner in CCS^{spt}). Our calculus CCS^{spt} does not
 1178 have a generic operator for sequential composition, like CCS and unlike Esterel. This
 1179 is not necessary, because sequential composition can be coded in CCS by action prefixing
 1180 and parallel composition. More precisely, a sequential composition $P;Q$ is obtained as
 1181 a parallel composition $([P]_t | t:t.Q) \setminus \{t\}$ where $[P]_t$ is the process P , modified so when
 1182 it terminates, then the termination signal \bar{t} is sent. The signal t will then trigger the
 1183 sequentially downstream process Q . The operation $[P]_t$ can be implemented by a simple
 1184 (linear) syntactic transformation of P . In the same way we can detect termination of a
 1185 parallel composition $P = P_1 | P_2$. Since a parallel terminates if both threads terminate, we
 1186 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
 1187 s_2 are fresh “local” termination signals. The “termination detector” $T_{seq} \stackrel{def}{=} s_1:s_1.s_2:s_2.\bar{t}$
 1188 first waits for P_1 to terminate and then for P_2 , before it sends the signal \bar{t} indicating the
 1189 termination of the parallel composition. We could equally first wait for P_2 and then for
 1190 P_1 , if the difference is not observable from the outside. In cases where the termination
 1191 detector needs to permit both processes P_1 and P_2 to terminate in any order, however, we
 1192 need a different solution. We need to “wait concurrently” for s_1 or s_1 , not sequentially as in
 1193 T_{seq} . The obvious idea would be to use a *confluent sum* $T_{csum} \stackrel{def}{=} s_1:s_1.s_2:s_2.\bar{t} + s_2:s_2.s_1:s_1.\bar{t}$
 1194 as discussed in Milner [22] (Chap. 11.4). This does not work, because the sum must
 1195 explicitly unfold all possible orders in which the termination signals can arrive and thus
 1196 again creates an exponential descriptive complexity. Interestingly, in CCS^{spt} we can exploit
 1197 precedence scheduling to obtain a robust and compositional solution. Consider the process
 1198 $T_{par} \stackrel{def}{=} s.T_{par} + \bar{t}:\{s, \bar{t}\}$. It engages in an unbounded number of receptions on channel s
 1199 without changing its state. At the same time, it offers to send the termination signal \bar{t} ,
 1200 but only if the action s is not possible. Under the weak enabling strategy, the composition
 1201 $\bar{s} | \bar{s} | T$ will consume the two senders \bar{s} and only then engage in t . In this way, we can use
 1202 T_{par} to detect the *absence* of s which corresponds to the termination of the two environment
 1203 processes $\bar{s} | \bar{s}$, in any order. The signal s could be the shared termination signal of each
 1204 process P_1 and P_2 that is sent and consumed by T_{par} if and when the process terminates. If
 1205 we execute the parallel composition $P_1 | P_2 | T_{par}$ under constructive enabling, the termination

XX:32 Strong Priority and Determinacy in Timed CCS

1206 signal \bar{t} will not be sent until such time that both P_1 and P_2 have been able to send s to
 1207 T . From the scheduling point of view, the composition $P_1 | P_2$ acts like a counter that is
 1208 initialised to 2, representing the number of potential offerings of s initially in $P_1 | P_2$. It
 1209 gets decremented each time one of P_1 or P_2 terminates. For instance, $0 | P_2$ and $P_1 | 0$ have
 1210 one potential offering left, while $0 | 0$ has no more potential to send s . At this moment, the
 1211 counter has reached 0 and $0 | 0 | T_{par}$ can take the \bar{t} action and thereby send termination.
 1212 In sum, there are two important observations here: First, we find that precedences with
 1213 constructive enabling permits us to code sequential composition of Esterel in a compositional
 1214 fashion. Second, the coding of Milner's confluent sum T_{csum} is possible in CCS^{spt} with linear
 1215 descriptive complexity.

1216 **► Example 37 (On Multi-Cast and Constructive Enabling).** Suppose $S_0 = a:e.S_0 + e.S_1 +$
 1217 $\sigma:\{a, e\}.S_0$ is an Esterel Signal (Sec. 11) in its absent state, with abbreviations $e = emit$,
 1218 $a = abs$ and $p = pres$. Its initial action $S_0 \xrightarrow{a}_{\{e\}} S_0$ to communicate absence does not block
 1219 itself to permit multi-reader scenarios. A typical reader is of form $R = \bar{p}:\bar{p}.R_1 + \bar{a}:\{\bar{a}, \bar{p}\}.R_0$.
 1220 This is a choice: if the signal is present we execute R_1 , otherwise if it is absent we execute
 1221 R_0 . The two sender actions \bar{p} and \bar{a} are self-blocking, because these prefixes can only be
 1222 consumed once. In essence, what we want is that there can be many senders of the value
 1223 labels (readers) but at most one receiver (signal).

1224 Consider a reader $C = \bar{a}:\bar{a}.\bar{e}:\bar{e}.C_1$ that wants to emit the signal when it finds it absent.
 1225 The composition $S_0 | C$ is deterministic under strong enabling. $S_0 | C \xrightarrow{\tau}_{0|\bar{e}\{e,\bar{a}\}} S_0 | \bar{e}:\bar{e}.C_1$.
 1226 The blocking set contains two labels:

1227 **(1)** First, $e \in \{e, \bar{a}\}$ indicates priority, i.e., that we do not want another concurrent thread
 1228 sending \bar{e} , because the synchronisation for absence is based on the assumption that
 1229 there is no emission. Specifically, $S_0 | C | \bar{e} \xrightarrow{\tau}_{0|\bar{e}\{e,\bar{a}\}} S_0 | \bar{e}:\bar{e}.C_1 | \bar{e}$ will block because
 1230 $\{e, \bar{a}\} \cap \bar{a}^\tau(0 | 0 | \bar{e}) \neq \{\}$. Instead, the transition $S_0 | C | \bar{e} \xrightarrow{\tau}_{0|C|0\{\bar{e}\}} S_1 | C | 0$ which emits
 1231 the signal can proceed.

1232 **(2)** Second, $\bar{a} \in \{e, \bar{a}\}$ indicates that the sending prefix $\bar{a}:\bar{a}$ of the reader C (which has entered
 1233 into the synchronisation $\tau = a | \bar{a}$) must not be consumed by another competing receiver.
 1234 Specifically, $S_0 | C | a \xrightarrow{\tau}_{0|0|a\{e,\bar{a}\}} S_0 | \bar{e}:\bar{e}.C_1 | a$ will block.

1235 Now, take $S_0 | C | D$ where the signal S_0 is connected with two readers, where C is as above
 1236 and $D = \bar{a}:\bar{a}.\bar{e}:\bar{e}.D_1$ analogous. Both readers can go ahead and synchronise in reactions
 1237 $(\tau = a | \bar{a}) S_0 | C | D \xrightarrow{\tau}_{0|0|D\{e,\bar{a}\}} S_0 | \bar{e}:\bar{e}.C_1 | D$ and $S_0 | C | D \xrightarrow{\tau}_{0|C|0\{e,\bar{a}\}} S_0 | C | \bar{e}:\bar{e}.D_1$ which
 1238 are both strongly enabled since $\bar{a}^\tau(0 | 0 | D) = \{a\} = \bar{a}^\tau(0 | C | 0)$ which is disjoint from the
 1239 blocking set $\{e, \bar{a}\}$. This may seem ok, as we wanted to have multiple readers access the signal
 1240 on the same port a . However, now the emission of \bar{e} by the process that advanced first will
 1241 block the reading of absence by the other which stayed behind. In each case symmetrically,
 1242 there is only one (strongly enabled) transition left, viz. $S_0 | \bar{e}:\bar{e}.C_1 | D \xrightarrow{\tau}_{0|\bar{e}\{e,\bar{a}\}} S_1 | C_1 | D$
 1243 and $S_0 | C | \bar{e}:\bar{e}.D_1 \xrightarrow{\tau}_{0|C|0\{\bar{e}\}} S_1 | C | D_1$ where the final states $C_1 | D$ and $C | D_1$ of the readers
 1244 are behaviourally different. Thus, we have non-determinism while the confluence is lost.

1245 The core of the problem is that the concurrent contexts $0 | 0 | D$ and $0 | C | 0$, respectively, of
 1246 the above rendez-vous synchronisations and do not account for the fact that the signal S_0 can
 1247 be simultaneously reused by the other reader that is not involved in the transition. We could
 1248 account for this reuse by adding S_0 into the concurrent context as in $S_0 | C | D \xrightarrow{\tau}_{S_0|0|D\{e,\bar{a}\}}$
 1249 $S_0 | \bar{e}:\bar{e}.C_1 | D$. Now D can interact with S_0 in the concurrent context $S_0 | 0 | D$. The above
 1250 transition depends on the assumption that no other thread will send \bar{e} . This is not true,
 1251 because now $e \in \bar{a}^\tau(S_0 | 0 | D)$. Thus, putting the shared signal S_0 into the concurrent

1252 environment is enforcing determinism by blocking the strong enabling properties of the above
 1253 transitions. Unfortunately, the fix is too strong. The presence of S_0 in the concurrent context
 1254 would block even the single reader in $S_0 \mid \bar{a}:\bar{a}$. Specifically, the reduction $S_0 \mid \bar{a}:\bar{a} \xrightarrow{\tau}_{S_0 \mid 0} \{e,a\}$
 1255 $S_0 \mid 0$ would block, because $\bar{a} \in \bar{iA}(S_0) \subseteq \bar{iA}^\tau(S_0 \mid 0)$, and so the reduction is not strongly
 1256 enabled any more. Thus, our fix would not permit any reading of the signal S_0 , whatsoever.

1257 The solution is to tighten up the notion of strong enabling and close it under arbitrary non-
 1258 clock transitions in the concurrent environment rather than just the silent steps. Technically,
 1259 we define a larger set $\bar{iA}^*(E) \supseteq \bar{iA}^\tau(E)$ of potential labels (Def. 13) that closes under
 1260 *all* non-clock actions of E . In this case, we do find $e \in \{e, \bar{a}\} \cap \bar{iA}^*(0 \mid 0 \mid D) \neq \{\}$. In
 1261 this way, diverging reductions are no longer enabled. On the other hand, the reduction
 1262 $S_0 \mid C \xrightarrow{\tau}_{0 \mid 0} \{e,a\} S_0 \mid \bar{e}:\bar{e}.C_1$ remains enabled, because $\bar{iA}^*(0 \mid 0) = \{\}$. In general, single readers
 1263 $S_0 \mid C$ where C is an arbitrary sequential process, or multiple readers $S_0 \mid C \mid D$ where *at most*
 1264 *one* of C or D is writing, i.e., sending \bar{e} , remain enabled.

1265 The last two examples show the policy conformance of Esterel signals presented in Sec. 11
 1266 and another ABRO encoding using multi-cast.

1267 ▶ **Example 38** (On Esterel signals, reloaded). A signal has the sort $\mathcal{L}(S_v) = \{emit, pres, abs, \sigma\}$
 1268 and is conformant to the policy psig with $\text{psig} \Vdash emit \dashrightarrow abs$ and $\text{psig} \Vdash \ell \dashrightarrow \sigma$ for all
 1269 $\ell \in \{emit, pres, abs\}$.

1270 ▶ **Example 39** (On Fully Multi-Cast ABRO). To model full multi-cast ABRO in CCS^{spt} , we
 1271 could use sharable (i.e., non-self-blocking) prefixes on internal actions to show how they are
 1272 useful to model the control-flow of Esterel programs, specifically termination and reset.

1273 ■ The reset watchdog R can receive a reset signal r or silently move to state $\sigma.R$ where it
 1274 synchronises on the clock σ to repeat in the next macro-step. When a reset r is received by
 1275 the environment, then R executes the sequence $r:r.\bar{k}_1:\bar{k}_1.\bar{k}_2:\bar{k}_2.\bar{k}_4:\bar{k}_4.\bar{k}_5:\bar{k}_5.ABRO$ whereby
 1276 it sends a “kill” signal \bar{k}_i to each of the four threads A , B , T and O , in some arbitrary
 1277 but fixed order. When all kills have been delivered, the watchdog R will restart ABRO.
 1278 Until such time, there is no clock possible. The second transition $R \xrightarrow{\tau}_{0 \mid \{r\}} \sigma.R$ preempts
 1279 this reset choice but has lower priority as it is blocked by r . It is only taken when there
 1280 is no reset possible, i.e., when the reset signal r is absent. Note that with the preemption
 1281 of the the kill signals \bar{k}_i are removed from the potential, because $\bar{iA}^\tau(\sigma.R) = \{\}$, i.e., we
 1282 do not look past the clock prefix.

1283 ■ The A and B processes are analogous to each other with the role of a and b swapped.
 1284 Looking at A , we see that it has a choice of three actions, viz to receive a kill signal k_1 ,
 1285 an input signal a or perform the clock action σ to start the next macro step. The kill k_1
 1286 has highest priority, action a second and the clock can only go ahead if there is neither a
 1287 k_1 nor an a signal to be received. When there is no kill but an a signal, then the process
 1288 A sends an \bar{s} signal to indicate its termination. When the kill k_1 occurs, it enters into the
 1289 halt state 1 where it is inactive but engages in any number of clock ticks. Since this is the
 1290 neutral element for parallel composition, A essentially drops out of the picture. Note that
 1291 we cannot use the inactive 0 here, because then the terminated process A would block
 1292 the clock for all other threads. Finally, when there is neither a kill k_1 nor signal a , the A
 1293 process can synchronise on σ and repeat as A in the next macro-step. In this fashion, A
 1294 will patiently wait for a clock tick in which k_1 or a occur. If both occur, the kill takes
 1295 priority: For instance, the transition $A \mid \bar{k}_1.1 \mid \bar{a}.1 \xrightarrow{\tau}_{0 \mid \bar{a}.1} \{k_1\} 1 \mid 1 \mid \bar{a}.1$ is constructively
 1296 enabled while $A \mid \bar{k}_1.1 \mid \bar{a}.1 \xrightarrow{\tau}_{0 \mid \bar{k}_1.1 \mid 0} \{k_1, a\} 1 \mid \bar{k}_1.1 \mid 1$ is not enabled.

XX:34 Strong Priority and Determinacy in Timed CCS

1297 ■ The process T is the termination detector implementing the principle discussed above.
 1298 It forms a clock-patient, killable counter that waits for any possible reception of signal
 1299 s before it sends termination \bar{t} to process O . Note that T itself does not need to know
 1300 how many possible termination signals there are. It waits for the termination count
 1301 (implemented by iA^*) to reach zero.
 1302 ■ When process O receives the termination signal t it sends the output signal \bar{o} to the
 1303 environment and then halts. Alternatively, with highest priority, it can be killed by
 1304 reception of k_5 or engage in the clock σ if neither a kill k_5 nor a termination t is present.
 1305 The reduction $\tau = \bar{t}:\{k_4, \bar{t}, s\} | t:\{k_5, t\}$ between T and O will be blocked by precedence,
 1306 for as long as at least one of the processes A or B has a choice term that sends \bar{s} , i.e., as
 1307 long as $s \in i\bar{A}^\tau(A|B)$. Only when both prefix sequences $a:\{a, k_1\}.\bar{s}:\bar{s}.1$ and $b:\{b, k_2\}.\bar{s}:\bar{s}.1$
 1308 have been eliminated by reductions, then the termination can occur.
 1309 Note that we can integrate the termination test into the output process O if we let O loop
 1310 on consumption of s rather than T . Similarly, we can use only a single kill signal k shared
 1311 between A, B, O : The reset process R keeps sending \bar{k} unboundedly often, until all target
 1312 processes have received a copy and all receptions k have been consumed. At that point, it
 1313 takes a clock step to repeat $ABRO$ from the start. This yields the following simpler process:

$$\begin{aligned}
 1314 \quad ABRO &\stackrel{def}{=} \sigma.(R|ABO) \\
 1315 \quad ABO &\stackrel{def}{=} (A|B|O) \setminus \{s, t\} \\
 1316 \quad R &\stackrel{def}{=} r:r.R' + \tau:r.\sigma.R \\
 1317 \quad R' &\stackrel{def}{=} \bar{k}.R' + \tau:\bar{k}.ABRO \\
 1318 \quad A &\stackrel{def}{=} k:k.1 + a:\{k, a\}.\bar{s}:\bar{s}.1 + \sigma:\{k, a\}.A \\
 1319 \quad B &\stackrel{def}{=} k:k.1 + b:\{k, b\}.\bar{s}:\bar{s}.1 + \sigma:\{k, b\}.B \\
 1320 \quad O &\stackrel{def}{=} k:k.1 + s:k.O + t:\{k, t, s\}.\bar{o}:\bar{o}.1 + \sigma:\{k, s, t\}.O.
 \end{aligned}$$

C Auxiliary Results

1322 Here is a simple observation on the properties of $iA(P)$ and $iA^*(P)$ that will be used in the
 1323 proofs below.

1324 ► **Lemma 40.** *Let P, Q be processes:*

- 1325 (1) $P \xrightarrow[R]{\sigma} Q$ implies $R \equiv 0$
- 1326 (2) $\alpha \in iA(P)$ iff $P \xrightarrow{\alpha} Q$.
- 1327 (3) $iA(P) \subseteq iA(P|Q)$
- 1328 (4) If P is single-threaded, then $P \xrightarrow[R]{\alpha} Q$ implies $\alpha \in \mathcal{L}$, $R \equiv 0$ and $\tau \notin H$.

1329 **Proof.** All points follows by an inspection of the SOS rules, using the definition of $iA^\tau(P)$. ◀

1330 Lem. 40(1) means that clock actions always run in an empty concurrent environment. Observe
 1331 that Lem. 40(2) means that the set $iA(P)$ corresponds to the initial actions of P . The presence
 1332 of τ in the blocking set of a transition provides information about the set of initial actions of
 1333 a process. Specifically, in rule (Par_3) the presence of τ in the set $H = race(P|Q)$ indicates
 1334 a blocking situation arising from the priorities in P or Q conflicting with the rendezvous
 1335 actions ℓ or $\bar{\ell}$. Thus, the condition $\tau \notin H$ can be used to implement priority-based scheduling.
 1336 Lem. 40(3) implies that single-threaded processes are never blocked.

1337 ► **Lemma 41.** *Let P be an arbitrary process:*

- 1338 (1) $iA(P) \cap \mathcal{L} \subseteq iA^\tau(P) \subseteq iA^*(P)$

1339 (2) If $P \equiv Q$ then $iA^*(P) = iA^*(Q)$

1340 (3) If $P \xrightarrow{\alpha} Q$ for $\alpha \in \mathcal{R} \cup \{\tau\}$, then $iA^*(Q) \subseteq iA^*(P)$.

1341 **Proof.** All points follows by an inspection of the SOS rules, using the definition of $iA^\tau(P)$. ◀

1342 ▶ **Definition 42** (Pivot Policy). A policy π is called a pivot policy if $\bar{\pi} \leq \pi$. We call a process
1343 P pivotable if P conforms to a pivot policy.

1344 ▶ **Proposition 43.** A policy $\pi = (L, \dashrightarrow)$ is a pivot policy if $\bar{L} \subseteq L$ and for all $\ell_1, \ell_2 \in L$
1345 with $\ell_1 \neq \ell_2$ we have $\pi \Vdash \ell_1 \diamond \ell_2$ or $\pi \Vdash \bar{\ell}_1 \diamond \bar{\ell}_2$.

1346 **Proof.** Let π be a pivot policy. Since the alphabet of $\bar{\pi}$ is \bar{L} and the alphabet of π is L the
1347 assumption $\bar{\pi} \leq \pi$ implies $\bar{L} \subseteq L$ by definition of \leq . Given labels $\ell_1, \ell_2 \in L$ with $\ell_1 \neq \ell_2$
1348 suppose $\pi \not\Vdash \ell_1 \diamond \ell_2$. Hence $\pi \Vdash \ell_1 \dashrightarrow \ell_2$ or $\pi \Vdash \ell_2 \dashrightarrow \ell_1$. Consider the first case, i.e.,
1349 $\pi \Vdash \ell_1 \dashrightarrow \ell_2$. Since $\bar{L} \subseteq L$ and $L = \bar{L}$ it follows that also $L \subseteq \bar{L}$, i.e., the labels ℓ_i are also
1350 in the alphabet of $\bar{\pi}$. But then by definition of the ordering and the fact that $\ell_i = \bar{\ell}_i$, the
1351 assumption $\bar{\pi} \leq \pi$ implies that $\bar{\pi} \Vdash \bar{\ell}_1 \dashrightarrow \bar{\ell}_2$. Now, the definition of the dual policy means
1352 that $\pi \not\Vdash \bar{\ell}_1 \dashrightarrow \bar{\ell}_2$ and $\pi \not\Vdash \bar{\ell}_2 \dashrightarrow \bar{\ell}_1$. This is the same as $\pi \Vdash \bar{\ell}_1 \diamond \bar{\ell}_2$ as desired.

1353 Suppose the properties (1) and (2) of the proposition hold for a policy π . We claim that
1354 $\bar{\pi} \leq \pi$. The inclusion of alphabets is by assumption (1). Then, suppose $\pi \Vdash \bar{\ell}_1 \dashrightarrow \bar{\ell}_2$ which
1355 implies $\pi \not\Vdash \bar{\ell}_1 \diamond \bar{\ell}_2$. If $\ell_1 = \ell_2$ we have $\bar{\pi} \Vdash \bar{\ell}_1 \dashrightarrow \bar{\ell}_2$ directly by definition. So, let $\ell_1 \neq \ell_2$.
1356 Because of $\bar{L} \subseteq L$ and (2) this gives us $\pi \Vdash \ell_1 \diamond \ell_2$ considering again that $\ell_i = \bar{\ell}_i$. But
1357 $\pi \Vdash \ell_1 \diamond \ell_2$ is the same as $\bar{\pi} \Vdash \bar{\ell}_1 \dashrightarrow \bar{\ell}_2$ by definition. ◀

1358 ▶ **Proposition 44.** Suppose P is input-scheduled and $P \xrightarrow[\bar{R}]{\alpha} Q$ for some α, H, R, Q . Then:

- 1359 1. If $\alpha \in \bar{\mathcal{A}}$ then $H \subseteq \{\alpha, \tau\}$
1360 2. If $\alpha \in \mathcal{A} \cup \mathcal{C} \cup \{\tau\}$ then $H \subseteq \mathcal{A} \cup \mathcal{C} \cup \{\tau\}$.

1361 **Proof.** Obvious by definition of conformance and the construction of π_{is} . ◀

1362 ▶ **Lemma 45.** Let $P_1 : \pi$ and $P_2 : \pi$ be coherent for the same pivot policy π and $\ell \in \mathcal{L}$ such
1363 that $P_1 \xrightarrow[\bar{R}_1]{\ell} P'_1$ and $P_2 \xrightarrow[\bar{R}_2]{\ell} P'_2$. Then, $H_2 \cap \bar{iA}(P_1) \subseteq \{\bar{\ell}\}$ and $H_1 \cap \bar{iA}(P_2) \subseteq \{\ell\}$.

1364 **Proof.** Let $\ell \in \mathcal{L}$ and the transitions be given as in the statement of the Lemma. Suppose
1365 $\alpha \in H_2$ and $\alpha \in \bar{iA}(P_1) \subseteq \mathcal{L}$, in particular $\alpha \neq \tau$. The former means $\pi \Vdash \alpha \dashrightarrow \bar{\ell}$ by
1366 Def. 16(1). Since π is pivot we must thus have $\pi \Vdash \bar{\alpha} \diamond \ell$, specifically $\pi \not\Vdash \bar{\alpha} \dashrightarrow \ell$, i.e.,
1367 $\bar{\alpha} \notin H_1$, again by Def. 16(1). But since $\bar{\alpha} \in iA(P_1)$ there is a transition $P_1 \xrightarrow[\bar{E}]{\bar{\alpha}} Q$. Since
1368 also $\pi \not\Vdash \ell \dashrightarrow \bar{\alpha}$, it follows $\ell \notin H$ by Def. 16(1). Now both $\bar{\alpha} \notin H_1$ and $\ell \notin H$ mean that
1369 the c-actions $\ell : H_1[R_1]$ and $\bar{\alpha} : H[E]$ are interference-free. Note that $\{\ell, \bar{\alpha}\} \subseteq \mathcal{L}$. Thus, the
1370 coherence Def. 16 for P_1 , implies $\bar{\alpha} = \ell$ or $\bar{\alpha} \in iA(R_1)$. The latter is impossible, because
1371 $H_2 \cap \bar{iA}(R_1) \subseteq H_2 \cap \bar{iA}(R_1 | R_2) = \{\}$ by assumption. Therefore we have $\alpha = \ell$ as desired.
1372 The inclusion $H_1 \cap \bar{iA}(P_2) \subseteq \{\ell\}$ is obtained by a symmetric argument. ◀

1373 ▶ **Lemma 46.** Let $P : \pi$ be coherent for a pivot policy π . Then, for every reduction $P \xrightarrow[\bar{E}]{\tau} P'$,
1374 if $H \cap \bar{iA}(E) = \{\}$ then $\tau \notin H$.

1375 **Proof.** Let P be coherent for pivotable π and a reduction $P \xrightarrow[\bar{E}]{\tau} P'$ such that $H \cap \bar{iA}(E) = \{\}$
1376 be given. Suppose, by contraposition, $\tau \in H$. Since there are no action prefixes $\tau.P'$ in

XX:36 Strong Priority and Determinacy in Timed CCS

1377 CCS^{spt} , the reduction must arise from a synchronisation of two threads P_1 and P_2 inside P ,
 1378 via rule Par_3 , i.e.,

$$1379 \quad \frac{P_1 \xrightarrow{R_1} P'_1 \quad P_2 \xrightarrow{R_2} P'_2 \quad B = \{\ell | \bar{\ell} \mid H_2 \cap \bar{\text{iA}}(P_1) \not\subseteq \{\bar{\ell}\} \text{ or } H_1 \cap \bar{\text{iA}}(P_2) \not\subseteq \{\bar{\ell}\}\}}{P_1 \mid P_2 \xrightarrow{R_1 \mid R_2} P'_1 \mid P'_2} \quad (\text{Par}_3)$$

1380 for some $\ell \in \mathcal{L}$, where we may assume, by induction on the depth of the derivation, that
 1381 $\tau \notin H_1 \cup H_2$. The assumption $H \cap \bar{\text{iA}}(E) = \{\}$ implies that $(H_1 \cup H_2) \cap \bar{\text{iA}}(R_1 \mid R_2) = \{\}$
 1382 at the point of (Par_3) . We must show that the synchronisation action τ cannot enter
 1383 into the blocking set, i.e., $\tau = \ell | \bar{\ell} \notin B$, i.e., which is the same as $H_2 \cap \bar{\text{iA}}(P_1) \subseteq \{\bar{\ell}\}$ and
 1384 $H_1 \cap \bar{\text{iA}}(P_2) \subseteq \{\bar{\ell}\}$. The sub-expressions $P_1 : \pi'$ and $P_2 : \pi'$ are also coherent for a common
 1385 pivot policy π' . This means we can apply Lem. 45 and 46 to show $B = \{\}$ and we are
 1386 done. \blacktriangleleft

D Proofs

1387 **Of Prop. 15.** First, non-interference of $\alpha_i : H_i[E_i \mid R]$ means $\alpha_i \notin H_{3-i}$ by assumption. Since
 1388 $H'_i \subseteq H_i$, this implies also $\alpha_i \notin H'_{3-i}$. Second, suppose that $\alpha_{3-i} = \tau$ for some $i \in \{1, 2\}$. Then,
 1389 non-interference of $\alpha_1 : H_1[E_1 \mid R]$ and $\alpha_2 : H_2[E_2 \mid R]$ gives $\tau \notin H_i$ and $H_i \cap \bar{\text{iA}}^*(E_i \mid R) = \{\}$.
 1390 Since by Lem. 40, $\bar{\text{iA}}^*(E_i) \subseteq \bar{\text{iA}}^*(E_i \mid R)$, the inclusions $H'_i \subseteq H_i$ preserve this property, i.e.,
 1391 $\tau \notin H'_i$ and $H'_i \cap \bar{\text{iA}}^*(E_i) \subseteq H_i \cap \bar{\text{iA}}^*(E_i \mid R) = \{\}$. \blacktriangleleft

1393 **Of Thm. 17.** Let P be coherent and Q a derivative with c-enabled reductions $Q \xrightarrow{R_1} Q_1$ and
 1394 $Q \xrightarrow{R_2} Q_2$. If $Q_1 \equiv Q_2$ we are done immediately. Suppose $Q_1 \not\equiv Q_2$. Then c-enabling
 1395 implies $\tau \notin H_i$ as well as $H_i \cap \bar{\text{iA}}^*(R_i) = \{\}$, which implies $H_i \cap (\bar{\text{iA}}^*(R_i) \cup \{\tau\}) = \{\}$. But
 1396 then the c-actions $\tau : H_1[R_1]$ and $\tau : H_2[R_2]$ are interference-free, whence coherence Def. 16(2)
 1397 implies there exist H'_i and processes R'_i, Q' such that $Q_1 \xrightarrow{R'_1} Q'$ and $Q_2 \xrightarrow{R'_2} Q'$
 1398 and $R_1 \rightsquigarrow R'_1$ and $R_2 \rightsquigarrow R'_2$ with $H'_i \subseteq H_i$. Finally, observe that $\bar{\text{iA}}^*(R'_i) \subseteq \bar{\text{iA}}^*(R_i)$
 1399 by Lem. 41 from which we infer $H'_i \cap (\bar{\text{iA}}^*(R'_i) \cup \{\tau\}) \subseteq H_i \cap (\bar{\text{iA}}^*(R_i) \cup \{\tau\})$. Thus, the
 1400 reconverging reductions $Q_i \xrightarrow{R'_{3-i}} Q'$ are again c-enabled. This was to be shown. \blacktriangleleft

1401 **Of Prop. 19.** The first part of the statement follows directly from Def. 16(2): The c-actions
 1402 $\ell : H_i[R_i]$ are trivially interference-free. Now if $P_1 \not\equiv P_2$, then coherence gives us a strong
 1403 environment shifts implying $\ell \in \text{iA}(R_1)$. The second part is because by Lem. 40(1) we have
 1404 $R_i = 0$ and thus $\text{iA}(R_i) = \{\}$ when $\ell \in C$. \blacktriangleleft

1405 **Of Lem. 23.** Consider an admissible transition of $P_1 \mid P_2$ where P_1 and P_2 are single-threaded.
 1406 Firstly, by Lem. 40, each of the single-threaded processes can only generate c-actions $\ell_i : H_i[R_i]$
 1407 where $\tau \notin H_i$ and $R_i = 0$. Such transitions are always c-enabled for trivial reasons. Further,
 1408 any rendez-vous synchronisation of such an $\ell_1 : H_1[R_1]$ and $\ell_2 : H_2[R_2]$ generates a c-action
 1409 $\tau : (H_1 \cup H_2 \cup B)[R_1 \mid R_2]$ with $R_1 \mid R_2 = 0$. Because of Lem. 45, the set

$$1410 \quad B = \{\tau \mid H_1 \cap \bar{\text{iA}}(P_2) \subseteq \{\bar{\ell}\} \text{ or } H_2 \cap \bar{\text{iA}}(P_1) \subseteq \{\bar{\ell}\}\}$$

1411 must be empty. Thus, the reduction $\ell_1 \mid \ell_2$ is c-enabled, too. \blacktriangleleft

1412 **Of Prop. 24.** The proof proceeds by contradiction. Let $P : \pi$ for pivot policy π . Suppose
 1413 $\ell \in \mathcal{L}$, $\ell \neq \sigma$ and $P \xrightarrow{\sigma}_H P_1$ and $P \xrightarrow{\ell}_F P_2$ and $P \xrightarrow{\bar{\ell}}_G P_2$. By Prop. 18 we infer $\sigma \in N$ or
 1414 $\ell \in H$, i.e., Conformance Def. 21 we have $\pi \Vdash \sigma \dashrightarrow \ell$ or $\pi \Vdash \ell \dashrightarrow \sigma$. Hence, $\pi \not\Vdash \sigma \diamond \ell$. For

1415 the same reason we have $\pi \not\Vdash \sigma \diamond \bar{\ell}$. However, this contradicts the pivot property of π , which
 1416 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
 1417 rendez-vous synchronisation inside P , i.e., there is $\ell \in \mathcal{R}$ with $\ell \in \text{iA}(P)$ and $\bar{\ell} \in \text{iA}(P)$. But
 1418 then $\sigma \in \text{iA}(P)$ is impossible as we have just seen. ◀

1419 D.1 Stop and Prefixes

1420 ▶ **Proposition 47.** *The process 0 is coherent for any policy, i.e., $0 : \pi$ for all π .*

1421 **Proof.** The terminating process 0 is locally coherent for trivial reasons, simply because it
 1422 does not offer any transitions at all. ◀

1423 ▶ **Proposition 48.** *Let process Q be coherent for π . Then, for every action $\ell \in \mathcal{R}$ the prefix*
 1424 *expression $\ell:H.Q$ is coherent for π , if $\ell \in H \subseteq \{\ell' \mid \pi \Vdash \ell' \dashrightarrow \ell\}$.*

1425 **Proof.** A prefix expression $P = \ell:H.Q$ generates only a single transition by rule (*Act*), so
 1426 that the assumption $P \xrightarrow[E_1]{\alpha_1} Q_1$ and $P \xrightarrow[E_2]{\alpha_2} Q_2$ implies $\alpha_i = \ell$, $H_i = H$, $E_i = 0$ and
 1427 $Q_i = Q$. It is easy to see that $\ell:H.Q$ conforms to π if $H \subseteq \{\beta \mid \pi \Vdash \beta \dashrightarrow \alpha\}$. Since
 1428 $\alpha_i = \ell \in H = H_i$ and $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ nothing needs to be proved. Finally, note that the only
 1429 immediate derivative of $\alpha:H.Q$ is Q which is coherent for π by assumption. ◀

1430 We will later see (Prop. 52) that a prefix $\ell:H.Q$ for $\ell \in \mathcal{R}$ can very well be coherent even if
 1431 $\ell \notin H$. Here we note that for clock prefixes, the blocking set is only constrained by the policy.

1432 ▶ **Proposition 49.** *If Q is coherent for π and clock $\sigma \in \mathcal{C}$, then the prefix expression $\sigma:H.Q$*
 1433 *is locally coherent for π , too, if $H \subseteq \{\beta \mid \pi \Vdash \beta \dashrightarrow \sigma\}$.*

1434 **Proof.** The argument runs exactly as in case of Prop. 48. However, we do not need to
 1435 require condition $\sigma \in H$ because if $\alpha_i = \sigma$ then $\alpha_1 = \alpha_2$, $Q_1 = Q_2$ and $\{\alpha_1, \alpha_2\} \not\subseteq \mathcal{R}$. So, no
 1436 reconvergence is required. Again, we observe that the only immediate derivative of $\sigma:H.Q$ is
 1437 Q which is coherent for π by assumption. ◀

1438 D.2 Summation

1439 ▶ **Proposition 50.** *Let $P_1 : \pi_1$ and $P_2 : \pi_2$ be coherent and for all pairs of initial transitions*
 1440 *$P \xrightarrow[F_1]{\alpha_1} P_1$ 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 $P_1 + P_2$*
 1441 *is coherent for π if $\pi_1 \subseteq \pi$ and $\pi_2 \subseteq \pi$.*

1442 **Proof.** Let $P = Q + R$ be given with Q and R locally coherent for π_1 and π_2 , respectively
 1443 and the rest as in the statement of the proposition. Every transition of P is either a
 1444 transition of Q or of R via rule *Sum*. Conformance to π directly follows from the same
 1445 property of Q or R , respectively. The proof of coherence Def. 16 is straightforward exploiting
 1446 that two competing but non-interfering transitions must either be both from Q or both
 1447 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 $P_1 \neq P_2$ or
 1448 $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ 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-
 1449 free. Now if the two transitions of P are by (*Sum*₁) from Q and by (*Sum*₂) from R , then
 1450 $\alpha_1 \in \text{iA}(Q)$ and $\alpha_2 \in \text{iA}(R)$. But then by assumption, $\alpha_1 \neq \alpha_2$ and $\alpha_1 \in H_2$ or $\alpha_2 \in H_1$,
 1451 which implies that $\alpha_1:H_1[F_1]$ and $\alpha_2:H_2[F_2]$ are not interference-free. This means, to prove
 1452 coherence Def. 16(2) we may assume that both transitions of P are either by (*Sum*₁) from
 1453 Q or both by (*Sum*₂) from R . Suppose both reductions of $P = Q + R$ are generated by
 1454 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)

XX:38 Strong Priority and Determinacy in Timed CCS

1455 are $Q + R \xrightarrow{F_1} Q_1$ and $Q + R \xrightarrow{F_2} Q_2$. Under the given assumptions, the reconverging
 1456 transitions of coherence Def. 16(2) are obtained directly from coherence of Q , by induction.
 1457 The case that both transitions are from R by rule (Sum_2) is symmetrical. Finally, the
 1458 immediate derivatives P_1 and P_2 of $P_1 + P_2$ are coherent for π , because they are coherent
 1459 for π_i by assumption and thus also for the common extension $\pi_i \subseteq \pi$. ◀

1460 D.3 Parallel

1461 ▶ **Lemma 51.** *Let $Q : \pi$ and $R : \pi$ be such that $\pi \setminus \mathcal{R}$ is a pivot policy. Then $(Q | R) : \pi$.*

1462 **Proof.** Let $P = Q | R$ such that both Q and R are coherent for π where $\pi \setminus \mathcal{R}$ is a pivot policy.
 1463 For conformance consider a transition $P \xrightarrow{E} P'$ with $\ell \in \mathcal{L}$ and $\ell' \in H$. First suppose $\ell \notin \mathcal{C}$.
 1464 Then this transition is not a synchronisation but a transition of either one of the sub-processes
 1465 Q by (Par_1) or P by (Par_2) with the same blocking set H . In either case, we thus use
 1466 coherence $Q : \pi$ or $R : \pi$ to conclude $\pi \Vdash \ell' \dashrightarrow \ell$. If $\ell = \sigma$, then the transition in question
 1467 is a synchronisation of Q and R via rule (Par_3) $Q | R \xrightarrow{\sigma} Q_2 | R_2$ with $E = 0$,
 1468 $H = H_2 \cup N_2 \cup B_2$ and $P' = Q_2 | R_2$ generated from transitions $Q \xrightarrow{\sigma} Q_2$ and $R \xrightarrow{\sigma} R_2$.
 1469 and $B_2 = \{\ell | \bar{\ell} \mid H_2 \cap \bar{iA}(R) \not\subseteq \{\sigma\} \text{ or } N_2 \cap \bar{iA}(Q) \not\subseteq \{\sigma\}\}$. By Lem. 45 we have $B_2 = \{\}$.
 1470 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
 1471 $\pi \Vdash \ell' \dashrightarrow \ell$ by conformance Def. 21 from $Q : \pi$ or $R : \pi$.

1472 In the sequel, we tackle coherence Def. 16. First we observe that the immediate derivatives
 1473 of $Q | R$ are always parallel compositions $Q' | R'$ of derivatives of Q and R . Thus, we may
 1474 assume that the immediate derivatives $Q' | R'$ are coherent for π because P and Q coherence is
 1475 preserved under transitions (i.e., by co-induction). Now suppose $P \xrightarrow{E_1} P_1$ and $P \xrightarrow{E_2} P_2$
 1476 such that $\alpha_1 \neq \alpha_2$ or $P_1 \neq P_2$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_i \notin H_{3-i}$. Moreover, the c-actions
 1477 $\alpha_1 : H_1[E_1]$ and $\alpha_2 : H_2[E_2]$ are interference-free. We argue reconvergence by case analysis on
 1478 the rules that generate these transitions. These could be (Par_1) , (Par_2) or (Par_3) . First
 1479 note that by Lem. 45 we can drop the consideration of the synchronisation action $\ell | \bar{\ell}$ for the
 1480 blocking set in (Par_3) altogether:

$$1481 \frac{P \xrightarrow{R_1} P' \quad Q \xrightarrow{R_2} Q'}{P | Q \xrightarrow{R_1 | R_2} P' | Q'} \quad (Par_3)$$

1482 We will use the name (Par_{3a}) to refer to the instance of (Par_3) with $\ell \in \mathcal{R}$ and the name
 1483 $(Par_{3\sigma})$ for the instance with $\ell \in \mathcal{C}$ and we proceed by case analysis.

1484 ■ $\{(Par_1), (Par_2)\} \diamond (Par_{3a})$. We start with the case where one of the reductions to P_i is non-
 1485 synchronising by (Par_1) or (Par_2) and the second reduction to P_{3-i} is a synchronisation
 1486 obtained by (Par_{3a}) . By symmetry, it suffices to consider the case that the non-
 1487 synchronising transition is by (Par_1) from Q and the synchronising transition by (Par_{3a})
 1488 from R , i.e., $\alpha_1 \notin \mathcal{C}$ and $\alpha_2 = \ell | \bar{\ell}$ and $Q | R \xrightarrow{F_1} Q_1 | R$ and $Q | R \xrightarrow{F_2} Q_2 | R_2$
 1489 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 $\ell \in \mathcal{R}$
 1490 is a rendez-vous action such that $Q \xrightarrow{F_1} Q_1$ and $Q \xrightarrow{F_2} Q_2$ and $R \xrightarrow{G_2} R_2$. We may
 1491 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
 1492 since $\alpha_2 = \tau$. Further, suppose the c-actions $\alpha_1 : H_1[F_1 | R]$ and $\ell | \bar{\ell} : (H'_2 \cup N_2)[F_2 | G_2]$
 1493 are non-interfering. Observe that $\{\alpha_1, \ell | \bar{\ell}\} \not\subseteq \mathcal{R}$ and also $\{\alpha_1, \ell | \bar{\ell}\} \cap \mathcal{C} = \{\}$. Thus
 1494 only a weak environment shift is needed. We wish to exploit coherence of Q . The first

1495 observation is that since $\alpha_2 = \tau$, the assumptions on non-interference Def. 14(2) implies
 1496 that $H_1 \cap \bar{\mathbf{A}}^*(F_1 | R) = \{\}$. Since $\ell = \bar{\ell} \in \bar{\mathbf{A}}(R) \cap \mathcal{L} \subseteq \bar{\mathbf{A}}^*(F_1 | R)$ by Lem. 40 and
 1497 Lem. 41, this implies that $\ell \notin H_1$ up front. Next, if $\alpha_1 \neq \ell | \bar{\ell}$, non-interference Def. 14(1)
 1498 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
 1499 by non-interference Def. 14(2) we have the non-interference equation

$$1500 \quad (H'_2 \cup N_2) \cap (\bar{\mathbf{A}}^*(F_2 | G_2) \cup \{\tau\}) = H_2 \cap (\bar{\mathbf{A}}^*(E_2) \cup \{\tau\}) = \{\}$$

1501 from which $\alpha_1 = \tau \notin H'_2$ follows. This settles the first part of the non-interference
 1502 property Def. 14(1) for the diverging transitions out of Q .

1503 For the second part of non-interference, suppose $\alpha_1 = \tau$. Then the non-interference
 1504 assumption implies that $H'_2 \cap \bar{\mathbf{A}}^*(F_2 | G_2) = \{\}$. Now since $\bar{\mathbf{A}}^*(F_2) \subseteq \bar{\mathbf{A}}^*(F_2 | G_2)$ we
 1505 conclude from this that $H'_2 \cap \bar{\mathbf{A}}^*(F_2) = \{\}$. This completes the proof that the c-actions
 1506 $\alpha_1 : H_1[F_1]$ and $\ell : H'_2[F_2]$ must be non-interfering in all cases.

1507 Note that we always have $\alpha_1 \neq \ell$ or $\{\alpha_1, \ell\} \subseteq \mathcal{R}$, and in the latter case also $\alpha_1 \notin$
 1508 H'_2 and $\ell \notin H_1$ from the above. Hence, we can use coherence Def. 16 of Q and
 1509 obtain processes F'_1, F'_2, Q' with reconverging transitions $Q_1 \xrightarrow{F'_2}^{F'_2} Q'$ and $Q_2 \xrightarrow{F'_1}^{\alpha_1} Q'$
 1510 such that $H''_2 \subseteq H'_2$ and $H'_1 \subseteq H_1$. We now invoke (Par_{3a}) and (Par_1) to obtain
 1511 reconverging transitions $Q_1 | R \xrightarrow{F'_2 | G_2}^{\ell | \bar{\ell}} H''_2 \cup N_2 Q' | R_2$ and $Q_2 | R_2 \xrightarrow{F'_1 | R_2}^{\alpha_1} H'_1 Q' | R_2$ such
 1512 that $H'_1 \subseteq H_1$ and $H''_2 \cup N_2 \subseteq H'_2 \cup N_2$. Regarding the concurrent environments, if
 1513 $\alpha_1 \in \mathcal{R}$ then the coherence of Q guarantees that $F_1 \xrightarrow{\ell} F'_1$ and $F_2 \xrightarrow{\alpha_1} F'_2$ from which we
 1514 infer $F_2 | G_2 \xrightarrow{\alpha_1} F'_2 | G_2$ and $F_1 | R \xrightarrow{\ell | \bar{\ell}} F'_1 | R_2$. If $\alpha_1 = \tau$ the above guarantee from the

1515 coherence of Q is weaker: We only have $F_1 \rightsquigarrow^{\ell} F'_1$ and $F_2 \rightsquigarrow^{\alpha_1} F'_2$. In any case we obtain
 1516 the environment shifts $F_2 | G_2 \rightsquigarrow^{\alpha_1} F'_2 | G_2$ and $F_1 | R \rightsquigarrow^{\ell | \bar{\ell}} F'_1 | R_2$, as required.

1517 ■ $(Par_1) \diamond (Par_2)$: Suppose the two non-interfering transitions of $P = Q | R$ are by (Par_1)
 1518 from Q and by (Par_2) from R . Thus, we are looking at transitions $Q \xrightarrow{F_1}^{\alpha_1} Q_1$ and $R \xrightarrow{F_2}^{\alpha_2} R_2$
 1519 R_2 for $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R} \cup \{\tau\}$ combined via (Par_1) and (Par_2) , respectively, to generate
 1520 $Q | R \xrightarrow{F_1 | R}^{\alpha_1} Q_1 | R$ and $Q | R \xrightarrow{Q | F_2}^{\alpha_2} Q | R_2$ with $E_1 = F_1 | R$ and $E_2 = Q | F_2$. Then
 1521 we can directly construct reconverging transitions, applying (Par_2) and (Par_1) , respectively,
 1522 for $Q_1 | R \xrightarrow{Q_1 | F_2}^{\alpha_2} Q_1 | R_2$ and $Q | R_2 \xrightarrow{F_1 | R_2}^{\alpha_1} Q_1 | R_2$. Obviously, $Q_1 | R_2$ reduces to
 1523 $Q_1 | R_2$ and by (Par_1) and (Par_2) we also have the strong environment shifts $F_1 | R \xrightarrow{\alpha_2}$
 1524 $F_1 | R_2$ and $Q | F_2 \xrightarrow{\alpha_1} Q_1 | F_2$ which completes the claim for we have, in particular,
 1525 $F_1 | R \rightsquigarrow^{\alpha_2} F_1 | R_2$ and $Q | F_2 \rightsquigarrow^{\alpha_1} Q_1 | F_2$. Notice that in this case we do not need
 1526 to assume that Q or R are coherent, i.e., the assumption that the c-actions $\alpha_i : H_i[E_i]$ are
 1527 interference-free.

1528 ■ $(Par_i) \diamond (Par_i)$ for $i \in \{1, 2\}$: Suppose both non-interfering and diverging reductions
 1529 are by (Par_1) from Q (or symmetrically both from R by (Par_2)) $Q | R \xrightarrow{F_1 | R}^{\alpha_1} Q_1 | R$
 1530 and $Q | R \xrightarrow{F_2 | R}^{\alpha_2} Q_2 | R$ with $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R} \cup \{\tau\}$ and $P_i = Q_i | R$, $E_i = F_i | R$, where
 1531 $Q \xrightarrow{F_1}^{\alpha_1} Q_1$ and $Q \xrightarrow{F_2}^{\alpha_2} Q_2$. with $\alpha_1 \neq \alpha_2$ or $Q_1 | R \neq Q_2 | R$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and
 1532 $\alpha_i \notin H_{3-i}$. This, in particular, implies $Q_1 \neq Q_2$. Further, assume non-interference of
 1533 $\alpha_i : H_i[F_i | R]$. We obtain non-interference of $\alpha_i : H_i[F_i]$ by Prop. 15. We can therefore
 1534 apply coherence Def. 16 of Q and obtain processes F'_1, F'_2 and Q' with reconverging
 1535 transitions $Q_1 \xrightarrow{F'_2}^{\alpha_2} H'_2 Q'$ and $Q_2 \xrightarrow{F'_1}^{\alpha_1} H'_1 Q'$ with $H'_i \subseteq H_i$ and such that

$$1536 \quad F_1 \rightsquigarrow^{\alpha_2} F'_1 \text{ and } F_2 \rightsquigarrow^{\alpha_1} F'_2 \tag{1}$$

1537 and if $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 \neq Q_2$, more strongly

$$1538 \quad F_1 \xrightarrow{\alpha_2} F'_1 \text{ and } F_2 \xrightarrow{\alpha_1} F'_2. \tag{2}$$

1539 Reapplying (Par_1) we construct the transitions $Q_1 | R \xrightarrow{F'_2 | R}^{H'_2} Q' | R$ and $Q_2 | R \xrightarrow{F'_1 | R}^{H'_1}$
1540 $Q' | R$. Obviously, by (Par_1) and (Par_2) we also obtain transitions $F_1 | R \xrightarrow{\alpha_1} F'_1 | R$ and
1541 $F_2 | R \xrightarrow{\alpha_2} F'_2 | R$ from (1) or, more strongly, $F_1 | R \xrightarrow{\alpha_2} F'_1 | R$ and $F_2 | R \xrightarrow{\alpha_1} F'_2 | R$ from (2)
1542 if $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 | R \neq Q_2 | R$.

1543 ■ $(Par_{3a}) \diamond (Par_{3a})$: This is the most interesting case in which we are going to exploit
1544 the assumption that Q and R are coherent for the same pivot policy π . Without loss of
1545 generality we assume $P = Q | R$ and both the reductions $Q | R \xrightarrow{E_i}^{H_i \cup N_i} Q_i | R_i$ with
1546 $P_i = Q_i | R_i$ are τ -actions generated by the communication rule (Par_3) , i.e. $\alpha_1 = \ell_1 | \bar{\ell}_1 =$
1547 $\tau = \ell_2 | \bar{\ell}_2 = \alpha_2$ for actions $\ell_i \in \mathcal{R}$. Since $\alpha_1 = \alpha_2$ and $\{\alpha_1, \alpha_2\} = \{\tau\} \not\subseteq \mathcal{R}$ we only need
1548 to prove confluence the case that $P_1 \neq P_2$, i.e., $Q_1 \neq Q_2$ or $R_1 \neq R_2$. Moreover, we only
1549 need a weak environment shift, because $\{\alpha_1, \alpha_2\} = \{\tau\} \not\subseteq \mathcal{R}$ and $\{\alpha_1, \alpha_2\} \cap \mathcal{C} = \{\}$. Also,
1550 non-interference Def. 14(2) means that $(H_i \cup N_i) \cap (\bar{iA}^*(E_i) \cup \{\tau\}) = \{\}$, i.e.,
1551 $H_i \cap (\bar{iA}^*(E_i) \cup \{\tau\}) = \{\}$ and $N_i \cap (\bar{iA}^*(E_i) \cup \{\tau\}) = \{\}$. (3)

1552 Thus, we are looking at transitions $Q \xrightarrow{F_i}^{\ell_i} Q_i$ and $R \xrightarrow{G_i}^{\bar{\ell}_i} R_i$. We claim that the two
1553 c-actions $\ell_i : H_i[F_i]$ of Q , and likewise the two c-actions $\bar{\ell}_i : N_i[G_i]$ of R , are interference-
1554 free. Note that since $\ell_i \neq \tau$, we only need to consider the first part of Def. 14(1).
1555 We first show that at least one of these pairs of c-actions must be interference free,
1556 because of coherence and conformance to π where $\pi \setminus \mathcal{R}$ is a pivot policy. It will then
1557 transpire that the other must be interference-free, too, because of the assumption (3).
1558 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
1559 trivial reasons. We distinguish two cases depending on whether ℓ_1 and ℓ_2 are identical
1560 or not. So, suppose $\ell_1 \neq \ell_2$. Both Q and R are coherent for the same policy π and
1561 $\pi \setminus \mathcal{R}$ is a pivot policy. Thus, $\pi \Vdash \ell_1 \diamond \ell_2$ or $\pi \Vdash \bar{\ell}_1 \diamond \bar{\ell}_2$, by Def. 42. Hence, $\ell_i \notin H_{3-i}$
1562 or $\bar{\ell}_i \notin N_{3-i}$ by conformance Def. 21, which means that at least one of the pairs of
1563 c-actions $\ell_i : H_i[F_i]$ of Q or the two c-actions $\bar{\ell}_i : N_i[G_i]$ must be interference-free by policy.
1564 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-
1565 free the other must be, too, and so we can close the diamond for both Q and R . By
1566 symmetry, it suffices to run the argument in case that the c-actions $\ell_i : H_i[F_i]$ of Q are
1567 non-interfering. Then, we have $\{\ell_1, \ell_2\} \subseteq \mathcal{R}$ and $\ell_i \notin H_{3-i}$. Hence, we invoke coherence
1568 Def. 16 of Q obtaining processes F'_1 and F'_2 with transitions (strong environment shift)
1569 $F_1 \xrightarrow{\ell_2} F'_1$ and $F_2 \xrightarrow{\ell_1} F'_2$ and a process Q' with transitions $Q_1 \xrightarrow{F'_2}^{H'_2} Q' | R$ and $Q_2 \xrightarrow{F'_1}^{H'_1} Q' | R$.
1570 such that $H'_i \subseteq H_i$. This means $\ell_i \in iA(F_{3-i}) \subseteq iA(F_{3-i} | G_{3-i}) = iA(E_{3-i}) \subseteq iA^*(E_{3-i})$
1571 and therefore $\bar{\ell}_i \notin N_{3-i}$ because of (3). Thus, as claimed, the c-actions $\bar{\ell}_i : N_i[G_i]$ are
1572 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
1573 Def. 16 for R from which we obtain processes G'_1 and G'_2 with transitions (again, a
1574 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
1575 $R_1 \xrightarrow{G'_2}^{N'_2} R'$ and $R_2 \xrightarrow{G'_1}^{N'_1} R'$. with $N'_i \subseteq N_i$. Finally, we invoke (Par_3) to obtain the re-
1576 converging reductions $Q_1 | R_1 \xrightarrow{F'_2 | G'_2}^{H'_2 \cup N'_2} Q' | R'$ and $Q_2 | R_2 \xrightarrow{F'_1 | G'_1}^{H'_1 \cup N'_1} Q' | R'$ with
1577 $H'_i \cup N'_i \subseteq H_i \cup N_i$, and also $F_i | G_i \xrightarrow{\ell_{3-i} | \bar{\ell}_{3-i}} F'_i | G'_i$ as required, since this implies
1578 the weaker form $F_i | G_i \xrightarrow{\tau} F'_i | G'_i$ of environment shift. It remains to consider the
1579 case that $\ell_1 = \ell_2 = \ell$. By assumption $P_1 \neq P_2$ and so we must have $Q_1 \neq Q_2$ or
1580 $R_1 \neq R_2$. Suppose $Q_1 \neq Q_2$. If $R_1 \neq R_2$ we apply a symmetric argument with the
1581 role of ℓ_i and $\bar{\ell}_i$ interchanged. As observed above, the c-actions $\ell_i : H_i[F_i]$ are trivially
1582 interference-free according to Def. 14, because $\ell_1 = \ell_2$ and $\ell_i \neq \tau$. Then, by coherence
1583 Def. 16 applied to Q we conclude that there are processes F'_1 and F'_2 with transitions
1584 $F_1 \xrightarrow{\ell} F'_1$ and $F_2 \xrightarrow{\ell} F'_2$ and a process Q' with transitions (the strong environment shift

1585 arises here because we have $\ell \in \mathcal{R}$ and $Q_1 \neq Q_2$) $Q_1 \xrightarrow{F'_2} Q'$ and $Q_2 \xrightarrow{F'_1} Q'$. such
 1586 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
 1587 $N_{3-i} \cap \bar{\text{iA}}^*(E_{3-i}) = \{\}$ as per (3). If $R_1 = R_2$ then for each $i \in \{1, 2\}$, by coherence
 1588 Def. 16 (as $\bar{\ell} \notin N_{3-i} \cup \mathcal{C}$) there is a transition $R_i \xrightarrow{G'_{3-i}} R'$ with $N'_i \subseteq N_i$ and $G_i = G'_i$
 1589 or $G_i \xrightarrow{\bar{\ell}} G'_i$. Using these transitions we now recombine $Q_i | R_i \xrightarrow{F'_{3-i} | G'_{3-i}} Q' | R'$
 1590 with $H'_i \cup N'_i \subseteq H_i \cup N_i$. Note that if $G_i = G'_i$ then $F_i | G_i \xrightarrow{\ell} F'_i | G'_i$ or if $G_i \xrightarrow{\bar{\ell}} G'_i$
 1591 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
 1592 $R_1 \neq R_2$ we can apply coherence Def 16 to R in the same way and close the diamond
 1593 with processes G'_1 and G'_2 and transitions (again, the strong environment shift arises here
 1594 because we have $\bar{\ell} \in \mathcal{R}$ and $R_1 \neq R_2$) $G_1 \xrightarrow{\bar{\ell}} G'_1$ and $G_2 \xrightarrow{\bar{\ell}} G'_2$ as well as a process R' with
 1595 $R_1 \xrightarrow{G'_2} R'$ and $R_2 \xrightarrow{G'_1} R'$. with $N'_i \subseteq N_i$. Finally, we invoke (Par_3) to obtain the re-
 1596 converging reductions $Q_1 | R_1 \xrightarrow{F'_2 | G'_2} Q' | R'$ and $Q_2 | R_2 \xrightarrow{F'_1 | G'_1} Q' | R'$ with
 1597 $H'_i \cup N'_i \subseteq H_i \cup N_i$, and also $F_i | G_i \xrightarrow{\ell | \bar{\ell}} F'_i | G'_i$ which also implies $F_i | G_i \xrightarrow{\ell | \bar{\ell}} F'_i | G'_i$ as
 1598 required.

1599 ■ (Par_{3a}) \diamond ($Par_{3\sigma}$) and, by symmetry, the induction case ($Par_{3\sigma}$) \diamond (Par_{3a}): We assume
 1600 $P = Q | R$ and the non-interfering, diverging reductions $Q | R \xrightarrow{E_1} Q_1 | R_1$
 1601 and $Q | R \xrightarrow{\sigma} Q_2 | R_2$ for $\ell \in \mathcal{L}$, $\sigma \in \mathcal{C}$, with $P_i = Q_i | R_i$ and $E_i = F_i | G_i$
 1602 are generated by instances (Par_{3a}) and ($Par_{3\sigma}$) of the communication rules, respectively.
 1603 These transitions arise from $Q \xrightarrow{F_1} Q_1$ and $R \xrightarrow{G_1} R_1$ and $Q \xrightarrow{\sigma} Q_2$ and $R \xrightarrow{\sigma} R_2$.
 1604 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
 1605 non-interfering. Note that $\ell \neq \sigma$ and $\bar{\ell} \neq \sigma$. First, observe that by Lem. 45, $B_1 = \{\} = B_2$.
 1606 Also, $\ell | \bar{\ell} = \tau \neq \sigma$ and so the non-interference assumption means that $\sigma \notin H_1 \cup N_1 \cup B_1$
 1607 as well as $\tau \notin H_2 \cup N_2 \cup B_2$. The other part of the non-interference, from Def. 14(1),
 1608 does not provide any extra information.

1609 We claim that $\ell \in H_2$. By contradiction, if $\ell \notin H_2$ then, since $\sigma \notin H_1$, the two c-actions
 1610 $\ell:H_1[F_1]$ and $\sigma:H_2[0]$ of Q would be interference-free according to Def. 14. But then,
 1611 since $\{\ell, \sigma\} \cap \mathcal{C} \neq \{\}$, coherence Def. 16 applied to Q would imply $\sigma \in \text{iA}(F_1)$ and $\ell \in \text{iA}(0)$.
 1612 The latter, however, is impossible. Thus, as claimed, $\ell \in H_2$. Likewise, from $\sigma \notin N_1$ we
 1613 derive $\bar{\ell} \in N_2$ in the same fashion.

1614 Now we invoke the fact that $\tau \notin B_2$, i.e. $H_2 \cap \bar{\text{iA}}(R) \subseteq \{\sigma\}$ and $N_2 \cap \bar{\text{iA}}(Q) \subseteq \{\sigma\}$. But
 1615 $\bar{\ell} \in \text{iA}(R)$ and $\ell \in \text{iA}(Q)$, whence we must have $\ell \notin H_2$ and $\bar{\ell} \notin N_2$. This is a contradiction.
 1616 Hence, transitions obtained by rules Par_{3a} and $Par_{3\sigma}$ are never interference-free.

1617 ■ $\{(Par_1), (Par_2)\} \diamond (Par_{3\sigma})$: As the representative case, we assume $P = Q | R$ and the non-
 1618 interfering, diverging reductions are $Q | R \xrightarrow{\alpha_1} Q_1 | R$ and $Q | R \xrightarrow{\sigma} Q_2 | R_2$
 1619 with $\alpha_1 \in \mathcal{R} \cup \{\tau\}$ so that $P_1 = Q_1 | R$ and $P_2 = Q_2 | R_2$ are generated by the rule (Par_1)
 1620 and ($Par_{3\sigma}$), respectively, from transitions $Q \xrightarrow{\alpha_1} Q_1$ and $Q \xrightarrow{\sigma} Q_2$ and $R \xrightarrow{\sigma} R_2$.
 1621 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-
 1622 free. First, Def. 14(1) implies $\alpha_1 \notin H_2 \cup N_2 \cup B_2$ and $\sigma \notin H_1$. Further, if $\alpha_1 = \tau$ then
 1623 $(H_2 \cup N_2 \cup B_2) \cap (\text{iA}^*(0) \cup \{\tau\}) = \{\}$, which in particular means $H_2 \cap (\text{iA}^*(0) \cup \{\tau\}) =$
 1624 $\{\}$. Therefore, the c-actions $\alpha_1:H_1[F_1]$ and $\sigma:H_2[0]$ are interference-free. Note that
 1625 $\{\alpha_1, \sigma\} \cap \mathcal{C} \neq \{\}$. Hence we can exploit coherence Def. 16 for Q , obtaining the stronger
 1626 form of environment shift. But this would imply $\alpha_1 \in \text{iA}(0)$ which is impossible. So we
 1627 find that transitions obtained by rules Par_i and $Par_{3\sigma}$ are never interference-free in the
 1628 sense of coherence.

XX:42 Strong Priority and Determinacy in Timed CCS

1629 ■ $(Par_{3\sigma}) \diamond (Par_{3\sigma})$: We assume $P = Q | R$ and the non-interfering, diverging reductions
 1630 $Q | R \xrightarrow[\sigma_0]{\sigma_1}_{H_1 \cup N_1 \cup B_1} Q_1 | R_1$ and $Q | R \xrightarrow[\sigma_0]{\sigma_2}_{H_2 \cup N_2 \cup B_2} Q_2 | R_2$ with $P_i = Q_i | R_i$ and $E_i =$
 1631 $F_i | G_i$ are generated by the communication rule $(Par_{3\sigma})$ and $(Par_{3\sigma})$. Observing that
 1632 $\{\sigma_1, \sigma_2\} \subseteq \mathcal{R}$ is impossible, here, we may assume $\sigma_1 \neq \sigma_2$ or $Q_1 | R_1 \neq Q_2 | R_2$. This can
 1633 only hold true if $Q_1 \neq Q_2$ or $R_1 \neq R_2$. Furthermore, the two clock transitions of $Q_1 | Q_2$
 1634 can be assumed to be interference-free. From this it follows that the two underlying clock
 1635 transitions $Q \xrightarrow[\sigma_0]{\sigma_1}_{H_1} Q_1$ and $Q \xrightarrow[\sigma_0]{\sigma_2}_{H_2} Q_2$ and the likewise the transitions $R \xrightarrow[\sigma_0]{\sigma_1}_{N_1} R_1$ and
 1636 $R \xrightarrow[\sigma_0]{\sigma_2}_{N_2} R_2$ are interference-free. If $Q_1 \neq Q_2$ then we could apply coherence of Q and
 1637 obtain $\sigma \in \text{iA}(0)$, if $R_1 \neq R_2$ the same follows from coherence of R . This is impossible,
 1638 whence the case can be excluded. ◀

1640 D.4 Repetition

1641 For arbitrary continuation processes A , a regular prefix $a:a.A$ needs to be self-blocking to
 1642 be coherent. Such a prefix cannot receive from two senders, for $a:a.A | \bar{a} | \bar{a}$ will block under
 1643 weak enabling. However, if the receiver is willing to engage in action a repeatedly, say if
 1644 $A \stackrel{\text{def}}{=} a.A$ then we can lift the self-blocking. The process $A | \bar{a} | \bar{a}$ is coherent and will happily
 1645 consume both \bar{a} sender actions. Repetition is the universal way of implementing multi-sender
 1646 and multi-receiver scenarios. Note that we define a new operator, representing recursion via
 1647 parallel replication on the same channel ℓ , that can be easily encoded in the current syntax.

1648 ▶ **Definition 52** (Sequential Bang Prefix). *For every process P , action $\ell \in \mathcal{L}$ and $H \subseteq \text{Act}$,*
 1649 *let $\ell^*:H.P$ be the process defined by the SOS rule*

$$1650 \quad \overline{\ell^*:H.P \xrightarrow[\ell]{P}_H P} \quad (\text{Rep})$$

1651 *We use $\ell^*:H$ as an abbreviation for $\ell^*:H.0$ and ℓ^* for $\ell^*:\{\}\cdot 0$.*

1652 The purpose of repetition is to replicate input and output prefixes so they can be consumed
 1653 multiple times and by multiple threads. To see this, let us notice the difference in the
 1654 blocking sets of a regular output prefix $\bar{a}:\bar{a}.A$ and its repetition $\bar{a}^* .A$. In the former case, the
 1655 transition $\bar{a}:\bar{a}.A \xrightarrow[\{\bar{a}\}]{\bar{a}} A$ includes the output label in the blocking set $\{\bar{a}\}$, by (Act_1) . This
 1656 reflects the fact that the output prefix \bar{a} is fully consumed by the transition. Accordingly,
 1657 (strong/weak) enabling will block multiple concurrent receivers trying to access the label a
 1658 at the same time. Only a single thread can engage in the synchronisation. This is necessary,
 1659 since the \bar{a} prefix is consumed. The continuation process A may not offer output \bar{a} any
 1660 more, or if it does, it may be incongruent with A . Therefore, each synchronisation with one
 1661 receiver thread would preempt another concurrent receiver thread in getting access to \bar{a} . For
 1662 contrast, in a transition $\bar{a}^* .A \xrightarrow[\{\}\cdot A]{\bar{a}} A | \bar{a}^* .A$ generated from (Act_1) and (Rep) , the blocking
 1663 set is empty, and so does not prevent concurrent receivers. This is fine since the prefix is not
 1664 consumed but repeated in the continuation process $A | \bar{a}^* .A$.

1665 ▶ **Remark 53** (Why is the standard bang not good enough?). Note that the sequential
 1666 bang $\ell^*.A$ is not expressible as $\ell^*.A = (\ell.A)^*$ by the standard ‘bang’ operator P^* of
 1667 process algebra which satisfies the *false* (hence not present) structural equivalence $P^* \equiv$
 1668 $P | P^*$ in CCS^{spt} . The corresponding equivalence $\ell^*.A \cong \ell.A | \ell^*.A$ does not hold in
 1669 CCS^{spt} . Let us see why. Suppose the behaviour of ℓ^* is derived from (or identified with
 1670 that of) $\ell.A | \ell^*.A$. Then the initial action of $\ell^*.A$ would be $\ell.A | \ell^*.A \xrightarrow[\{\}\cdot A]{\ell} A | \ell^*.A$
 1671 generated by rules (Act_1) and (Par_1) . Notice that the concurrent environment $\ell^*.A =$
 1672 $\ell^*.A$ is not empty and that we have $\ell \in \text{iA}(\ell^*.A)$. Therefore, a parallel composition

1673 $\bar{\ell}:\bar{\ell}|\ell.A|\ell^*.A \xrightarrow[\ell^*.A]{\tau} \{\bar{\ell}\} A|\ell^*.A$ would block under (weak/strong) enabling because $\{\bar{\ell}\} \cap$
 1674 $\bar{iA}(\ell^*.A) \neq \{\}$. Thus $\ell.A|\ell^*.A$ would not be able to synchronise even with a single
 1675 (self-blocking) sender $\bar{\ell}:\bar{\ell}$. For contrast, the sequential bang prefix of Def. 52 can serve
 1676 arbitrarily many receivers $\bar{\ell}:\bar{\ell}|\bar{\ell}:\bar{\ell}|\ell^*.A \xrightarrow[\bar{\ell}:\bar{\ell}|A]{\tau} \{\bar{\ell}\} \bar{\ell}:\bar{\ell}|A|\ell^*.A \xrightarrow[A|A]{\tau} \{\bar{\ell}\} A|A|\ell^*.A$, provided
 1677 that $\{\bar{\ell}\} \cap \bar{iA}^*(A) = \{\bar{\ell}\} \cap \bar{iA}^*(A) = \{\}$. The problem is that the standard bang $\ell^*.A \equiv$
 1678 $\ell.A|\ell^*.A$ is a *parallel* repetition of label ℓ while the sequential bang $\ell^*.A$ offers all ℓ -labels
 1679 *sequentially* in a single thread. Only the payload A is offered in parallel. This corresponds to
 1680 an *async* operator that is triggered by label ℓ , then spawns a child thread A and repeats
 1681 itself in the main thread. ◀

1682 ▶ **Proposition 54.** *Let $P : \pi$ be coherent and $\pi \setminus \mathcal{R}$ a pivot policy. Then, for each $\ell \in \pi \setminus \mathcal{R}$,*
 1683 *the sequential bang prefix $\ell^*:H.P$ is coherent for π if $H \subseteq \{\ell' \mid \pi \Vdash \ell' \dashrightarrow \ell\}$.*

1684 **Proof.** By definition, the process $\ell^*:H.P$ offers only a single initial transition $\ell^*:H.P \xrightarrow{P} H$
 1685 $P|\ell^*:H.P$ which must be obtained by application of rule *Rep*. The conformance Def. 21
 1686 follows exactly as for action prefixes from $\ell \in \pi$ the assumption $H \subseteq \{\ell' \mid \pi \Vdash \ell' \dashrightarrow \ell\}$.
 1687 Since there are no diverging transitions for $\ell^*:H.P$ the only situation to consider for
 1688 confluence is that $\ell \notin H$. For reconvergence it suffices to observe that the continuation
 1689 process $P|\ell^*.P$ offers another transition engaging with ℓ and the same blocking set H :
 1690 $P|\ell^*:H.P \xrightarrow{P} H P|P|\ell^*:H.P$. By structural equivalence, $P \rightsquigarrow 0|P$. This weak environment
 1691 shift is enough to satisfy coherence Def. 16, because of the deterministic transition of a
 1692 non-clock. Finally, note that since P is coherent for π and coherence for pivot policies
 1693 closed under parallel composition (Prop. 51), we can assume that the continuation process
 1694 $P|\ell^*:H.P$ is conformant to π , by co-induction. ◀

1695 D.5 Hiding

1696 ▶ **Proposition 55.** *If Q is coherent for π and $\pi \not\vdash \sigma \dashrightarrow \ell$ for all $\sigma \in L$, then Q/L is coherent*
 1697 *for π .*

1698 **Proof.** It suffices to prove the proposition for the special case Q/σ of a single clock. We
 1699 recall the semantic rule for this case:

$$1700 \frac{P \xrightarrow[E]{\alpha} Q \quad H' = H - \{\sigma\}}{P/\sigma \xrightarrow[E/\sigma]{\alpha/\sigma} H' Q/\sigma} \text{ (Hide)}$$

1701 where $\sigma/\sigma = \tau$ and $\alpha/\sigma = \alpha$ if $\alpha \neq \sigma$. Let $P = Q/\sigma$ and the two non-interfering
 1702 and diverging transitions $P \xrightarrow[E_1]{\alpha_1/\sigma} P_1$ and $P \xrightarrow[E_2]{\alpha_2/\sigma} P_2$ arise by (*Hide*) from transitions
 1703 $Q \xrightarrow[F_1]{\alpha_1} N_1 Q_1$ and $Q \xrightarrow[F_2]{\alpha_2} N_2 Q_2$ with $E_i = F_i/\sigma$, $P_i = Q_i/\sigma$ and $H_i = N_i - \{\sigma\}$. For coherence
 1704 Def. 16 we assume that $\alpha_1/\sigma \neq \alpha_2/\sigma$ or $P_1 \neq P_2$, or $\{\alpha_1/\sigma, \alpha_2/\sigma\} \subseteq \mathcal{R}$ and $\alpha_i/\sigma \notin H_{3-i}$.
 1705 Further, let the c-actions $\alpha_i:H_i[E_i]$ be interference-free. Observe that the first case implies
 1706 $\alpha_1 \neq \alpha_2$ and the second case means $Q_1 \neq Q_2$. Moreover, because of the assumption that
 1707 $\pi \not\vdash \sigma \dashrightarrow \alpha_i$ we must have $\sigma \notin N_1 \cup N_2$. But this means $H_i = N_i$. First, suppose that
 1708 $\alpha_1 = \alpha_2 = \sigma$. But then the assumption $Q_1 \neq Q_2$ contradicts strong determinacy of clocks
 1709 (Proposition 19). Thus, the actions α_i cannot both be the clock σ that is hidden. The
 1710 case $\alpha_i = \sigma$ and $\alpha_{3-i} \in Act - \{\sigma\}$, for some fixed $i \in \{1, 2\}$ can also be excluded as follows:
 1711 By Prop. 18 the clock transition and the non-clock transition must interfere in Q , i.e.,
 1712 $\alpha_i \in N_{3-i}$ or $\alpha_{3-i} \in N_i$. The former $\sigma = \alpha_i \in N_{3-i}$ is impossible because of the above. The
 1713 latter is outright impossible since then $\alpha_{3-i}/\sigma = \alpha_{3-i} \in N_i = H_i$ which contradicts the
 1714 non-interference assumption on the transitions of P .

XX:44 Strong Priority and Determinacy in Timed CCS

1715 The only remaining case to handle is that both of the diverging transitions are by labels
1716 $\alpha_1, \alpha_2 \in Act - \{\sigma\}$ and $\alpha_i / \sigma = \alpha_i$ for both $i \in \{1, 2\}$. We claim that the c-actions $\alpha_i : N_i[F_i]$
1717 are interference-free. If $\alpha_1 \neq \alpha_2$ then non-interference assumption directly gives $\alpha_i \notin H_i = N_i$.
1718 Next, if for some $i \in \{1, 2\}$, $\alpha_i = \alpha_i / \sigma = \tau$, then the assumptions on P not only give $\tau \notin$
1719 $H_{3-i} = N_{3-i}$ but also $N_{3-i} \cap \bar{i}\bar{A}^*(F_{3-i} / \sigma) = H_{3-i} \cap \bar{i}\bar{A}^*(E_{3-i}) = \{\}$. Since by assumption on
1720 policy conformance $\sigma \notin N_{3-i}$ the latter implies $N_{3-i} \cap \bar{i}\bar{A}^*(F_{3-i}) = \{\}$. Now we have shown
1721 that the diverging transitions of Q are non-interfering we can use the coherence Def. 16 for Q by
1722 induction hypothesis and get F'_1, F'_2, Q' such that $Q_1 \xrightarrow{F'_2}_{N'_2} Q'$ and $Q_2 \xrightarrow{F'_1}_{N'_1} Q'$ with $F_1 \xrightarrow{\alpha_2}_{F'_2}$
1723 F'_1 and $F_2 \xrightarrow{\alpha_1}_{F'_1} F'_2$ such that $N'_i \subseteq N_i$. By applying (*Hide*) to these transitions, we obtain
1724 $P_1 \xrightarrow{F'_2/\sigma}_{H''_2} Q' / \sigma$ and $P_2 \xrightarrow{F'_1/\sigma}_{H''_1} Q' / \sigma$ where $H''_i = N'_i - \{\sigma\} \subseteq N_i - \{\sigma\} = H_i$. Note, by
1725 rule (*Hide*) we also have $F_1 / \sigma \xrightarrow{\alpha_2/\sigma}_{F'_1/\sigma}$ and $F_2 / \sigma \xrightarrow{\alpha_1/\sigma}_{F'_2/\sigma}$. In the special case that
1726 $\{\alpha_1 / \sigma, \alpha_2 / \sigma\} = \{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_1 \neq \alpha_2$ or $Q_1 / \sigma \neq Q_2 / \sigma$, or $\{\alpha_1 / \sigma, \alpha_2 / \sigma\} \cap \mathcal{C} =$
1727 $\{\alpha_1, \alpha_2\} \cap \mathcal{C} \neq \{\}$ we obtain the stronger context shifts $F_1 \xrightarrow{\alpha_2}_{F'_1}$ and $F_2 \xrightarrow{\alpha_1}_{F'_2}$ from which
1728 we infer $F_1 / \sigma \xrightarrow{\alpha_2/\sigma}_{F'_1/\sigma}$ and $F_2 / \sigma \xrightarrow{\alpha_1/\sigma}_{F'_2/\sigma}$. This completes the proof. \blacktriangleleft

1729 D.6 Restriction

1730 Let us next look at the restriction operator. The following definition is relevant for creating
1731 coherent processes under action restriction (App. D.6).

1732 **► Definition 56.** A policy π is called precedence-closed for $L \subseteq \mathcal{L}$ if $\ell_1 \in L$ and $\pi \Vdash \ell_1 \dashv\dashv \ell_2$
1733 implies $\ell_2 \in L$.

1734 Again, the fact that P is locally coherent does not imply that $P \setminus L$ is locally coherent.
1735 Consider the process $Q \stackrel{def}{=} (s + a:s | b.\bar{s}) \setminus s$ which is conformant and pivotable. The signal
1736 s blocks the action a by priority in the thread $s + a:s$. For the communication partner \bar{s}
1737 to become active, however, it needs the synchronisation with an external action \bar{b} . The
1738 transition $Q \xrightarrow{(1|b.\bar{s}) \setminus s} \{ (b.\bar{s}) \setminus s \}$ generated by (*Restr*) does not provide enough information
1739 in the blocking set H and environment E for us to be able to characterise the environments
1740 in which the action a is blocked. For instance, the parallel composition $Q | \bar{b}$ will internally
1741 set the sender \bar{s} free and thus block the a -transition. From the above transition the (*Par*₃)
1742 rule permits $Q | \bar{b}$ to offer the a -step. This is not right as it is internally blocked and does
1743 not commute with the $b | \bar{b}$ reduction. A simple solution to avoid such problems and preserve
1744 confluence is to force restricted signals so they do not block any visible actions. This
1745 restriction suffices in many cases.

1746 As we have noted above, CCS corresponds to the unlocked and (blocking) free processes of
1747 CCS^{spt} (Prop. 26). For these processes, admissible transitions and strongly enabled transitions
1748 coincide. The key result of Milner [22] (Chap. 11, Prop. 19) is that confluence is preserved
1749 by inaction 0, prefixing $\ell.P$ and *confluent composition* defined as $P |_L Q \stackrel{def}{=} (P | Q) \setminus L$ where
1750 $\mathcal{L}(P) \cap \mathcal{L}(Q) = \{\}$ and $\overline{\mathcal{L}(P)} \cap \mathcal{L}(Q) \subseteq L \cup \bar{L}$. We will show how confluence for this fragment
1751 of CCS processes follows from our more general results on coherence. From now on, for the
1752 rest of this section, we assume that all processes are unlocked, i.e., each prefix $\ell:H.Q$ has
1753 $H \cup \{\ell\} \subseteq \mathcal{A} \cup \bar{\mathcal{A}}$.

1754 Let us call a process P *discrete* if all prefixes occurring in P are of form $\ell:\{\ell\}.Q$, i.e.,
1755 they are self-blocking. Under strong enabling, a discrete process behaves like a CCS process
1756 with the restriction that each action ℓ only synchronises if there is a matching partner $\bar{\ell}$ in
1757 a *unique* other parallel thread. If the matching partner is not unique, then the scheduling
1758 blocks. For instance, $\ell:\ell | \bar{\ell}:\bar{\ell} | \bar{\ell}:\bar{\ell}$ blocks because there are two concurrent senders $\bar{\ell}$ matching

1759 the receiver ℓ . Now consider the fragment CCS^{cc} of discrete processes built from inaction
 1760 0 , self-blocking prefixes $\ell:\{\ell\}.Q$ and confluent composition. Milner's result says that the
 1761 processes in CCS^{cc} are confluent under admissible scheduling.

1762 To emulate Milner's result in our setting, we consider the *sort* $\mathcal{L}(P)$ of a process P as the
 1763 admissible actions of a *discrete* policy π_P with only reflexive precedences. In other words,
 1764 $\pi_P \Vdash \ell_1 \dashrightarrow \ell_2$ iff $\ell_1 = \ell_2$. It is easy to see that a discrete process always conforms to π_P .
 1765 Also, discrete policies are always pivot policies (Def. 42) and dependency-closed for all label
 1766 sets (Def. 56), as one shows easily. Thus, discrete processes, which are coherent for discrete
 1767 policies, fulfill all conditions of the preservation laws for inaction 0 (Prop. 47), self-blocking
 1768 action prefix $\ell:\{\ell\}.Q$ (Prop. 48), parallel composition $P|Q$ (Prop. 51) and restriction $P \setminus L$
 1769 (Prop. 57). As a consequence and in particular, all processes in CCS^{cc} are coherent under
 1770 strong enabling by our results.

1771 Our final observation now is that, for Milner's fragment CCS^{cc} , the transition semantics
 1772 under admissible and strong enabling, as well as the notions of confluence and coherence,
 1773 coincide. Firstly, if P is discrete then in every c-action $\ell:H[E]$ executed by a derivative Q of P
 1774 we must have $H = \{\ell\}$. Thus, the coherence Def. 16(1) becomes redundant since it is always
 1775 satisfied. Further, any two c-actions $\ell_i:H_i[E_i]$ of transitions $Q \xrightarrow{\ell_1}_{E_1} Q_1$ and $Q \xrightarrow{\ell_2}_{E_2} Q_2$ for
 1776 $\ell_1 \neq \ell_2$ or $Q_1 \neq Q_2$ are trivially interference-free. The confluent composition of Milner has
 1777 the further effect that for every c-action $\ell:H[E]$ we must have $\ell \notin \overline{iA}^*(E)$. This is proven by
 1778 induction on the structure of $P \in \text{CCS}^{\text{cc}}$. Further, the side condition of Par_3 becomes trivial,
 1779 and thus generally $\tau \notin H$. This is a result of the side-conditions of the confluent composition.
 1780 Hence, for CCS^{cc} , strong enabling coincides with plain admissibility CCS. Notice that the
 1781 class of discrete processes and the class processes coherent for discrete policies is larger than
 1782 CCS^{CW} . In particular, we can explain coherence of broadcast actions with repetitive prefixes.
 1783 Hence, our results are more general even for the restricted class of discrete behaviours. This
 1784 will be exploited with the examples discussed in a proper section.

1785 **► Proposition 57.** *If $Q : \pi$ and π is precedence-closed for $L \cup \bar{L}$, then $(Q \setminus L) : \pi - L$.*

1786 **Proof.** It suffices to prove the proposition for the special case $P = Q \setminus a$ for a single channel
 1787 name $a \in A$. Recall the rule (*Restr*) for this case:

$$1788 \frac{Q \xrightarrow{\alpha}_H R \quad \alpha \notin \{a, \bar{a}\} \quad H' = H - \{a, \bar{a}\}}{Q \setminus a \xrightarrow{\alpha}_{E \setminus a} R \setminus a} \text{ (Restr)}$$

1789 Consider a transition $Q \setminus a \xrightarrow{\ell}_{E \setminus a} R \setminus a$ derived from $Q \xrightarrow{\ell}_H Q'$ with $H' = H - \{a, \bar{a}\}$,
 1790 $\ell \notin \{a, \bar{a}\}$ and $\ell' \in H'$. Then $\ell' \notin \{a, \bar{a}\}$ and $\ell' \in H$. Conformance Def. 21 applied to Q
 1791 implies $\pi \Vdash \ell' \dashrightarrow \ell$. But then $\pi - a \Vdash \ell' \dashrightarrow \ell$, too, as both ℓ and ℓ' are distinct from a
 1792 and \bar{a} . This ensures conformance. If $P = Q \setminus a$ and the two non-interfering and diverging
 1793 transitions $P_2 \xleftarrow{\alpha_2}_{E_2} P \xrightarrow{\alpha_1}_{E_1} P_1$ arise from rule (*Restr*) then we have diverging transitions
 1794 $Q_2 \xleftarrow{\alpha_2}_{F_2} Q \xrightarrow{\alpha_1}_{F_1} Q_1$ with $H_i = N_i - \{a, \bar{a}\}$, $\alpha_i \notin \{a, \bar{a}\}$, where $E_i = F_i \setminus a$ and $P_i = Q_i \setminus a$.
 1795 We assume $\alpha_1 \neq \alpha_2$ or $P_1 \neq P_2$ or $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$ and $\alpha_i \notin H_{3-i}$. Further, suppose non-
 1796 interference of $\alpha_1:H_1[E_1]$ and $\alpha_2:H_2[E_2]$. Note that in case $P_1 \neq P_2$ we immediately have
 1797 $Q_1 \neq Q_2$. Next, notice that the premise of the (*Restr*) rule ensures $\alpha_i \notin \{a, \bar{a}\}$. It is easy to
 1798 see that this means $\{a, \bar{a}\} \cap N_i = \{\}$, i.e., $H_i = N_i$. By contraposition, suppose $\ell \in \{a, \bar{a}\}$
 1799 and $\ell \in N_i$. Then $\pi \Vdash \ell \dashrightarrow \alpha_i$. But π is precedence-closed for $\{a, \bar{a}\}$, so we would have
 1800 $\alpha_i \in \{a, \bar{a}\}$ which is a contradiction. So, $H_i = N_i$.

1801 All this means that to invoke coherence Def. 16 of the above Q is sufficient to show that
 1802 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

XX:46 Strong Priority and Determinacy in Timed CCS

1803 non-interference assumption on the transitions out of P we get $\alpha_i \notin H_{3-i} = N_{3-i}$. This
1804 is what we need for Def. 14(1) to show that the c-actions $\alpha_i:N_i[F_i]$ are interference-free.
1805 For the second part Def. 14(2) suppose $\alpha_{3-i} = \tau$ for some $i \in \{1, 2\}$. Then the assumed
1806 non-interference of the transitions out of P means $H_i \cap (\overline{iA}^*(E_i) \cup \{\tau\}) = \{\}$, which is
1807 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
1808 $\{a, \bar{a}\} \cap N_{3-i} = \{\}$ we have $N_i \cap \overline{iA}^*(F_i) \subseteq \mathcal{L} - \{a, \bar{a}\}$. We claim that the α_i -transitions of
1809 Q is enabled, specifically $N_i \cap \overline{iA}^*(F_i) = \{\}$ observing that $\tau \notin H_i = N_i$. By contraposition,
1810 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
1811 also $\beta \in \overline{iA}^*(F_i \setminus a)$, but this contradicts the above set-condition. Thus, we have shown
1812 that for all $i \in \{1, 2\}$, $\alpha_{3-i} = \tau$ implies $N_i \cap \overline{iA}^*(F_i) = \{\}$. This proves the c-actions
1813 $\alpha_i:N_i[F_i]$ out of Q are interference-free. Hence, we can use coherence of Q for the diverging
1814 transition and get $F'_1, F'_2, Q' Q_1 \xrightarrow{F'_2} N'_2 Q'$ and $Q_2 \xrightarrow{F'_1} N'_1 Q'$ with $F_1 \xrightarrow{\alpha_2} F'_1$ and $F_2 \xrightarrow{\alpha_1} F'_2$
1815 such that $N'_i \subseteq N_i$. We construct the required reconvergence by application of rule (*Restr*):
1816 $P_1 \xrightarrow{E_2} H'_2 Q' \setminus a$ and $P_2 \xrightarrow{E'_1} H'_1 Q' \setminus a$ with $F_1 \setminus a \xrightarrow{\alpha_2} F'_1 \setminus a$ and $F_2 \setminus a \xrightarrow{\alpha_1} F'_2 \setminus a$ with $E'_i =$
1817 $F'_i \setminus a$ and $H'_i = N'_i - \{a, \bar{a}\} \subseteq N_i - \{a, \bar{a}\} = H_i$. In the special case that $\{\alpha_1, \alpha_2\} \subseteq \mathcal{R}$
1818 and $\alpha_1 \neq \alpha_2$ or $P_1 \neq P_2$ or $\{\alpha_1, \alpha_2\} \cap \mathcal{C} \neq \{\}$ we can rely on the stronger context shifts
1819 $F_1 \xrightarrow{\alpha_2} F'_1$ and $F_2 \xrightarrow{\alpha_1} F'_2$ to conclude $F_1 \setminus a \xrightarrow{\alpha_2} F'_1 \setminus a$ and $F_2 \setminus a \xrightarrow{\alpha_1} F'_2 \setminus a$. This completes
1820 the proof. ◀