Coherence and Determinacy in CCS with Priorities (Synpatick)

M. Mendler* and L. Liquori

Synchron 2023, Kiel

*work begun while on visit at INRIA Sophia/UCA in May 2023, still in progress
Our Result: Generalise Milner’s determinacy results for CCS by strengthening theories of CCS with priorities (e.g., $\text{CCS}^{\text{CW}}$ [Camilleri & Winskel 1995], $\text{CCS}^{\text{Ph}}$ [Phillips 2001]):

- **Twistit 1**: replace “weak enabling” by “constructive enabling”
- **Twistit 2**: replace “confluence” by “coherence”
- **Twistit 3**: replace “sort” $\mathcal{L}(P)$ by “policy type” $\pi(P)$.

Our Objective: ...to ground the semantics (and thus essence) of

- **sequentially constructive Esterel** [von Hanxleden et. al.; DATE’2013, PLDI’14, Memocode’15], and more generally
- **deterministic shared objects** [Aguado et. al: ESOP 2018]

in the setting of Milner’s process algebra CCS.
Roadmap

1. CCS (Syntax & Operational Semantics)
2. Milner’s CCS Confluence Class
3. CCS with Priorities (Synpa\textsuperscript{\text{tick}})
4. Twistit I: Constructive Enabling
5. Twistit II: Coherence for Constructive Enabling
6. Twistit III: Precedence Policies and Preservation of Coherence
7. Conclusion
CCS (Syntax & Operational Semantics)
Basic CCS Terminology

Identifiers

- channel names \( a, b, c \in \mathcal{A} \)
- process names \( A \in \mathcal{I} \)

Action Labels

- (channel) co-names \( \overline{a}, \overline{b}, \overline{c} \in \overline{\mathcal{A}} \)
- (rendez-vous) action labels \( \ell \in \mathcal{L} \overset{\text{def}}{=} \mathcal{A} \cup \overline{\mathcal{A}} \) ("a input", "\overline{a} output")
- actions \( \alpha, \beta \in \text{Act} = \mathcal{L} \cup \{\tau\} \) where \( \tau \notin \mathcal{L} \) silent action

Synchronising Actions \((\ell \in \mathcal{L}, L \subseteq \mathcal{L})\)

- \( \ell | \overline{\ell} = \tau = \overline{\ell} | \ell \)
- \( \overline{L} = \{\overline{\ell} \mid \ell \in L\} \)
- \( \overline{\overline{\ell}} = \ell \)
Syntax of CCS

Process Expressions

\[ P, Q, R, S ::= \]
\[ 0 \quad \text{stop (inaction)} \]
\[ \ell.P \quad \text{action prefix (} \ell \in \mathcal{L} \text{)} \]
\[ P + Q \quad \text{choice} \]
\[ P | Q \quad \text{parallel composition} \]
\[ P \setminus L \quad \text{restriction (} L \subseteq A \text{)} \]
\[ A \quad \text{identifier (} A \in \mathcal{I} \text{)} \]

Definitional Equations \( A \overset{df}{=} P \)

Abbreviation We write \( \ell \) instead of \( \ell.0 \).

Free Names \( FN(P) \subseteq A \cup \mathcal{I} \) (process identifiers remain unbound).

A process \( P \) is well-formed if every identifier \( A \in FN(P) \cap \mathcal{I} \) has a definitional equation. \( \mathcal{L}(P) = FN(P) \cap \mathcal{L} \) is the sort of \( P \).
Operational Semantics of CCS

\[
\frac{\ell . P}{P \xrightarrow{\ell} P} \quad (Act) \quad \frac{P \xrightarrow{\alpha} P'}{P \xrightarrow{\alpha} P', A \overset{df}{=} P} \quad (Con)
\]

\[
\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \quad (Sum_{1,2}) \quad \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q} \quad (Par_{1,2})
\]

\[
\frac{P \xrightarrow{\ell} P', Q \xrightarrow{\bar{\ell}} Q'}{P \parallel Q \xrightarrow{\ell | \bar{\ell}} P' \parallel Q'} \quad (Par_3) \quad \ell | \bar{\ell} = \tau
\]

\[
\frac{P \xrightarrow{\alpha} Q, \alpha \notin L \cup \bar{L}}{P \\setminus L \xrightarrow{\alpha} Q \setminus L} \quad (Restr)
\]

Rules taken modulo structural congruence \( P \equiv Q \).
We are interested in uniqueness of normal forms under \( \tau \)-reductions.

- \( P \) is normal if there is no \( P' \) such that \( P \xrightarrow{\tau} P' \).
- Write \( P \xrightarrow{\varepsilon} Q \) if \( P \equiv Q \) or \( P \xrightarrow{\tau} P' \) and (inductively) \( P' \xrightarrow{\varepsilon} Q \).

**Church-Rosser**

A process \( P \) satisfies Church-Rosser (CR) if for every derivative \( Q \) of \( P \) and reductions \( Q \xrightarrow{\tau} Q_1 \) and \( Q \xrightarrow{\tau} Q_2 \) with \( Q_1 \equiv Q_2 \) there exist \( Q'_1 \) and \( Q'_2 \) with \( Q'_1 \equiv Q'_2 \) and \( Q_1 \xrightarrow{\tau} Q'_1 \) and \( Q_2 \xrightarrow{\tau} Q'_2 \).

**Determinacy**

If \( P \) satisfies CR then \( P \) is determinate: If \( P \xrightarrow{\varepsilon} P_1 \) and \( P \xrightarrow{\varepsilon} P_2 \) where \( P_i \) are normal, then \( P_1 \equiv P_2 \).
Example

Observation: CR is not preserved under parallel composition.

Write-once Store: $S \overset{df}{=} w.S' + \bar{r}.S$ and $S' \overset{df}{=} \bar{r}.S''$

- $\bar{r}, w$ “store-side” read (output) and write (input)
- $r, \bar{w}$ “program-side” read (input) and write (output)

Reader $\mid S \mid$ Reader
Church-Rosser

Reader $\mid S \mid$ Writer
not Church-Rosser
Milner’s CCS Confluence Class
Milner’s Notion of Confluence

The classical theory of CCS defines confluence as a strengthening of CR and proves preservation of confluence for restricted parallel composition.

Confluence

A process $P$ is (structurally) confluent if for every derivative $Q$ of $P$ and transitions $Q \xrightarrow{\alpha_1} Q_1$ and $Q \xrightarrow{\alpha_2} Q_2$ such that

- $\alpha_1 \neq \alpha_2$ or $Q_1 \equiv Q_2$,
- there exist $Q'_1 \equiv Q'_2$ with $Q_1 \xrightarrow{\alpha_2} Q'_1$ and $Q_2 \xrightarrow{\alpha_1} Q'_2$.

Observation: Confluence $\Rightarrow$ Church-Rosser

Milner’s Confluence Class

- Confluent composition is given as $P \mid_{\mathcal{L}} Q = (P \mid_{\mathcal{L}} Q) \setminus \mathcal{L}$ for $L \subseteq \mathcal{L}$ with $\mathcal{L}(P) \cap \mathcal{L}(Q) = \{\}$ and $\overline{\mathcal{L}(P)} \cap \mathcal{L}(Q) = L \cup \overline{L}$.
- If $P$ and $Q$ are confluent, then $P \mid_{\mathcal{L}} Q$ is confluent, too.
The Limits of Milner’s Confluence Class

- Memory access $S = w.S' + r.S$ is intrinsically not confluent
- Confluent composition $P |_L Q$ precludes sharing of labels

But sequentially constructive synchronous programming exploits non-confluence and sharing of labels for ...

- deterministic shared memory:
  $\llbracket Mem \parallel Write \parallel ReadA \parallel ReadB \rrbracket \approx S | (W\{(t | \bar{t})/0\} | t.R_A | t.R_B) \backslash t$

- multi-cast communication:
  $\llbracket emit a \parallel present a then A \parallel present a then B \rrbracket \approx \bar{a}.a | a.A | a.B$

- sequential composition with upstream concurrency:
  $\llbracket (await a \parallel await b ); emit o \rrbracket \approx (a.t | b.t | t.t.o) \backslash t$
CCS with Priorities (Synpa$\text{tick}$)
Syntax of Synpa\textsuperscript{tick}

Extended Process Expressions

\[
P, \ Q, \ R, \ S \ ::= \ 0 \quad \text{stop (inaction)} \quad | \quad \ell \cdot P \quad \text{action prefix (} \ell \in \mathcal{L} \text{)} \quad | \quad P + Q \quad \text{choice} \quad | \quad P \parallel Q \quad \text{parallel composition} \quad | \quad P\backslash L \quad \text{restriction (} L \subseteq \mathcal{A} \text{)} \quad | \quad A \quad \text{identifier (} A \in \mathcal{I} \text{)} \quad | \quad P:H \quad \text{precedence guard (} H \subseteq \mathcal{L} \text{)}
\]

Idea: \( P:H \approx "P \text{ unless the environment offers an alternative in } H" \).

Abbreviation: Instead of \((\alpha.P):H\) write \(\alpha:H.P\) and \(\ell:H\) for \(\ell:H.0\).
Strategic SOS Semantics

The Plot: Enrich the “unscheduled” SOS semantics à la CCS

\[ P \xrightarrow{\alpha} P' \] by priority annotations

\[ P \xrightarrow{\alpha}_R P' \]

where the contextual action (c-action) \( \alpha: H[R] \) has

- \( H \subseteq \text{Act} \) blocking set of actions that take precedence over \( \alpha \)
- \( R \) is the concurrent context of threads in \( P \) that compete with \( \alpha \).

The Roadmap: Confluence for Strategic Scheduling

- **Twistit I:** Define a \( \Phi \)-enabled constraint \( \Phi(R, H) \) on c-actions\( \alpha: H[R] \)
- **Twistit II:** Define \( \Phi \)-confluence for \( \Phi \)-enabled c-actions that implies Church-Rosser.
- **Twistit III:** Show that \( \Phi \)-confluence is preserved by composition under reasonable restrictions but permitting sharing and memory.
Extended Operational Semantics

**Accumulating Blocking Sets**

\[
P \xrightarrow{\alpha}^{H'}_R P' \\
\vdash_{R \cup H} P \xrightarrow{\alpha}^{H'}_R P'
\]

*(Prio)*

**Accumulating Concurrent Context**

\[
P \xrightarrow{\alpha}^H R P' \\
P \parallel Q \xrightarrow{\alpha}^H R P' \parallel Q
\]

*(Par1,2)*

**Evaluating Blocking Conditions**

\[
P \xrightarrow{\ell}^{H_1} R_1 P' \\
Q \xrightarrow{\ell}^{H_2} R_2 Q' \\
H = \{ \tau \mid H_2 \cap \overline{\text{iA}}(P) \not\subseteq \{ \ell \} \text{ or } H_1 \cap \overline{\text{iA}}(Q) \not\subseteq \{ \ell \} \}
\]

*(Par3)*

**Initial Actions:**

\[\text{iA}(P) \overset{\text{def}}{=} \{ \ell \mid \exists H, R, P'. P \xrightarrow{\ell}^R_H P' \} \subseteq \mathcal{L}\]
Weakly Enabled Transitions

A transition $P \xrightarrow{\alpha}_R H P'$ is weakly enabled if $H \cap (\overline{iA}(R) \cup \{\tau\}) = \{\}$. 

CCS$^\text{Ph}$ Processes

- CCS$^\text{Ph}$ is the fragment of Synpa$^\text{tick}$ such that all blocking occurs in prefixes $(\ell.R):H$ only, with $\ell \notin H$.
- If $P \in \text{CCS}^\text{Ph}$, then

$$P \xrightarrow{\alpha}_R H P' \text{ is weakly enabled}$$

iff $P \xrightarrow{\alpha}_H P'$ is derivable in the semantics of CCS$^\text{Ph}$ [Phillips 2001].
Examples - Write-before-Read

Write-before-read Store:
\[ S = w.S' + r:w.S \text{ and } S' = r.S'' \]

Concurrent Environment: \( E = \bar{r} | \bar{w} \)

- The transition ("read first")

\[
S \mid E \equiv (w.S' + r:w.S) \mid \bar{r} \mid \bar{w} \xrightarrow{\text{prec}} \{w\} \ S \mid 0 \mid \bar{w} \]

is not weakly enabled, since \( \{w\} \cap \bar{A}(0 \mid 0 \mid \bar{w}) = \{w\} \neq \{\} \).

- Problem: Weak enabling does not eliminate data races, instead we need...

- The transition ("write first")

\[
S \mid E \equiv (w.S' + r:w.S) \mid \bar{r} \mid \bar{w} \xrightarrow{\text{prec}} \{\} \ S' \mid \bar{r} \mid 0 \]

is weakly enabled, since \( \{\} \cap (\bar{A}(0 \mid \bar{r} \mid 0) \cup \{\tau\}) = \{\} \).
Twistit I

Constructive Enabling
Constructively Enabled Transitions

- $P \xrightarrow{\alpha}_H P'$ is c-enabled if $H \cap (\overline{iA}^*(R) \cup \{\tau\}) = \emptyset$.

Potential Actions

- The set $iA^*(R) \subseteq \mathcal{L}$ of potential actions is the smallest extension $iA(R) \subseteq iA^*(R)$ such that* if $R \xrightarrow{\alpha} R'$ then $iA^*(R') \subseteq iA^*(R)$.

Note:

- $H \cap (\overline{iA}^*(R) \cup \{\tau\}) = \emptyset$ reminds of Esterel’s Cannot Analysis.
- Every constructively enabled transition is also weakly enabled.

* $\alpha$ not a clock
Twistit II
Coherence for Constructive Enabling
Coherence

Independence

- Two c-actions $\alpha_1 : H_1[E_1]$ and $\alpha_2 : H_2[E_2]$ are independent if $\{\alpha_1, \alpha_2\} \neq \{\tau\}$ and both $\alpha_1 \not\in H_2$ and $\alpha_2 \not\in H_1$.

Coherence

- A process $P$ is (structurally) coherent if for all its derivatives $Q$ and c-enabled transitions

$$Q \xrightarrow{\alpha_1}_{E_1} H_1 Q_1 \quad \text{and} \quad Q \xrightarrow{\alpha_2}_{E_2} H_2 Q_2$$

where the c-actions $\alpha_i : H_i[E_i]$ are independent or $\alpha_1 = \alpha_2$ and $Q_1 \equiv Q_2$. Then, there exist $Q_1' \equiv Q_2'$ and c-enabled transitions

$$Q_1 \xrightarrow{\alpha_2}_{E_2} H_2' Q_1' \quad \text{and} \quad Q_2 \xrightarrow{\alpha_1}_{E_1} H_1' Q_2'.$$
Coherent Sharing and Memory

The following are not confluent in CCS but coherent in Synpatick:

Esterel Signal (pure temporary, no clock):

\[ S_0 \overset{df}{=} \overline{abs} \cdot emit \cdot S_0 + emit \cdot S_1 \]
\[ S_1 \overset{df}{=} \overline{pres} \cdot S_1 + emit \cdot S_1 \]

→ permits multiple programs on co-names \( emit, abs, pres \).

Esterel Programs \((H = \{pres, abs\})\)

- \([\text{present } S \text{ then } P \text{ else } Q] \approx \overline{pres} : H.P + \overline{abs} : H.Q\)
- \([\text{emit } S; P] \approx \overline{emit} : \overline{emit} . [P]\)
- \([\text{await } A || \text{await } B; P] \approx (\overline{pres}_A : \overline{pres}_A . t | \overline{pres}_B : \overline{pres}_B . t | t . t . P) \setminus t\)

→ assumes there is a single signal on co-names \( emit, abs, pres \).
Twistit III
Policies & Preservation of Coherence
Precedence Policy

Policies replace CCS’ notion of the sort $\mathcal{L}(P)$ of a process.

### Precedence Policy

- A precedence policy (p-policy) $\pi = (L, \rightarrow)$ is a relation $\rightarrow \subseteq L \times L$ on a set of labels $L \subseteq \mathcal{L}$.
- $P$ conforms to $\pi$ if for all its derivatives $Q$, if $Q \xrightarrow{R} H Q'$, then $\alpha \in L$ and $\forall \ell \in H. \ell \rightarrow \alpha$.
- The policy type of $P$ is the (set-theoretically) smallest p-policy $\pi(P)$ so that $P$ conforms to $\pi(P)$.

### Policy Type $\pi_{\text{sig}}$ of Esterel Signals and Programs

\[
\begin{array}{cccccc}
\text{abs} & \text{emit} & \text{pres} & \text{abs} & \text{emit} \\
\text{---} & \text{----} & \text{----} & \text{----} & \text{---}
\end{array}
\]
Pivot Policy

The p-policy $\pi_{\text{sig}}$ has a special property...

Pivot Policy

A p-policy $\pi = (L, \rightarrow)$ is a pivot policy if

- it is closed under co-names, $\overline{L} \subseteq L$
- “rendez-vous synchronisation on distinct channels do not interfere each other”

Main Theorem (Generalising Milner’s Confluence Class)

- Coherent processes are Church-Rosser for c-enabled reductions.
- If $P$ and $Q$ are coherent and conform to pivot policy $\pi$, then $P \parallel Q$ is coherent* and conforms to $\pi$.

* Since we do not need to restrict we permit sharing!
Conclusion
Our Result: Generalise Milner’s determinacy results for CCS in CCS with priorities (e.g., CCS$^{cw}$ [Camilleri & Winskel 1995], CCS$^{ph}$ [Phillips 2001]):

- “constructive enabling” rather than “weak enabling”
- “coherence” rather than “confluence”
- “policy type” $\pi(P)$ rather than “sort” $\mathcal{L}(P)$.

Now What? Adding clocks (CSP broadcast action) we can now
- express sequentially constructive Esterel, and more generally
- express deterministic shared objects [Aguado et. al. ESOP 2018]
- explore the algebraic theory of c-enabling in Synpa$^{tick}$.
Thank You for Your Attention!