Coherence and Determinacy in CCS with Priorities ($\text{Synpa}^{\text{tick}}$)

M. Mendler* and L. Liquori

Synchron 2023, Kiel

* work begun while on visit at INRIA Sophia/UCA in May 2023, still in progress

Introduction

Our Result: Generalise Milner's determinacy results for CCS by strengthening theories of CCS with priorities (e.g., CCS^{cw} [Camilleri & Winskel 1995], CCS^{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$^{\text{tick}}$)
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 $\ell \in \mathcal{L} \stackrel{def}{=} \mathcal{A} \cup \overline{\mathcal{A}}$ ("a input", "$\overline{a}$ output")
- actions $\alpha, \beta \in Act = \mathcal{L} \cup \{\tau\}$ 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

$$
\begin{array}{rcll}
P, Q, R, S & ::= & 0 & \text{stop (inaction)} \\
& \mid & \ell.P & \text{action prefix } (\ell \in \mathcal{L}) \\
& \mid & P + Q & \text{choice} \\
& \mid & P \mid Q & \text{parallel composition} \\
& \mid & P \backslash L & \text{restriction } (L \subseteq \mathcal{A}) \\
& \mid & A & \text{identifier } (A \in \mathcal{I})
\end{array}
$$

Definitional Equations $A \stackrel{df}{=} P$
Abbreviation We write ℓ instead of $\ell.0$.

Free Names $FN(P) \subseteq \mathcal{A} \cup \mathcal{I}$ (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

$$\frac{}{\ell.P \xrightarrow{\ell} P}\ (Act) \qquad \frac{P \xrightarrow{\alpha} P' \quad A \overset{df}{=} P}{A \xrightarrow{\alpha} P'}\ (Con)$$

$$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}\ (Sum_{1,2}) \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ (Par_{1,2})$$

$$\frac{P \xrightarrow{\ell} P' \quad Q \xrightarrow{\overline{\ell}} Q'}{P \mid Q \xrightarrow{\ell \mid \overline{\ell}} P' \mid Q'}\ (Par_3) \quad \ell \mid \overline{\ell} = \tau$$

$$\frac{P \xrightarrow{\alpha} Q \quad \alpha \notin L \cup \overline{L}}{P \backslash L \xrightarrow{\alpha} Q \backslash L}\ (Restr)$$

Rules taken modulo structural congruence $P \equiv Q$.

Church-Rosser & Determinacy

We are interested in uniqueness of normal forms under τ-reductions.

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

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 $Q_1 \not\equiv Q_2$ 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

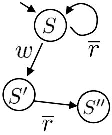
- If P satisfies CR then P is determinate: If $P \stackrel{\varepsilon}{\Rightarrow} P_1$ and $P \stackrel{\varepsilon}{\Rightarrow} P_2$ where P_i are normal, then $P_1 \equiv P_2$.

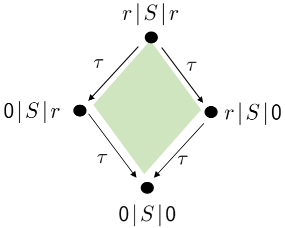
Example

Observation: CR is not preserved under parallel composition.

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

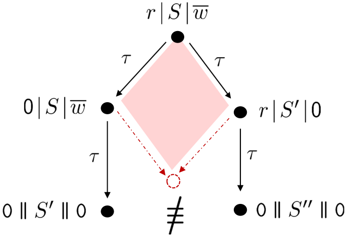
- $\overline{r}$, w “store-side” read (output) and write (input)
- r, $\overline{w}$ “program-side” read (input) and write (output)





Reader | S | Reader

Church-Rosser



Reader | S | 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 $Q_1 \not\equiv Q_2$,
- 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 $P \,|_L\, Q = (P \,|\, Q) \backslash L$ 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 $P \,|_L\, Q$ is confluent, too.*

The Limits of Milner's Confluence Class

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

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

- deterministic shared memory:
  $[\![\text{Mem} \,||\, \text{Write} \,||\, \text{ReadA} \,||\, \text{ReadB}]\!] \approx S \mid (W\{(\bar{t} \mid \bar{t})/0\} \mid t.R_A \mid t.R_B)\backslash t$
- multi-cast communication:
  $[\![\text{emit a} \,||\, \text{present a then A} \,||\, \text{present a then B}]\!] \approx \bar{a}.\bar{a} \mid a.A \mid a.B$
- sequential composition with upstream concurrency:
  $[\![(\text{await a} \,||\, \text{await b}); \text{emit o}]\!] \approx (a.\bar{t} \mid b.\bar{t} \mid t.t.\bar{o})\backslash t$

CCS with Priorities (Synpa^{tick})

Syntax of Synpa$^{\text{tick}}$

Extended Process Expressions

P, Q, R, S	$::=$	0	stop (inaction)
	$\mid$	$\ell.P$	action prefix ($\ell \in \mathcal{L}$)
	$\mid$	$P + Q$	choice
	$\mid$	$P \mid Q$	parallel composition
	$\mid$	$P \backslash L$	restriction ($L \subseteq \mathcal{A}$)
	$\mid$	A	identifier ($A \in \mathcal{I}$)
	$\mid$	$P{:}H$	precedence guard $(H \subseteq \mathcal{L})$

Idea: $P{:}H \approx$ "P 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 á la CCS

$$P \xrightarrow{\alpha} P' \quad \text{by priority annotations} \quad P \xrightarrow[R]{\alpha}_H P'$$

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

- $H \subseteq Act$ blocking set of actions that take precedence over α
- R is the concurrent context of threads in P that compete with α.

The Roadmap: Confluence for Strategic Scheduling

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

Extended Operational Semantics

Accumulating Blocking Sets

$$\frac{P \xrightarrow[R]{\alpha}_{H'} P'}{P{:}H \xrightarrow[R]{\alpha}_{H' \cup H} P'} \; (Prio)$$

Accumulating Concurrent Context

$$\frac{P \xrightarrow[R]{\alpha}_{H} P'}{P \mid Q \xrightarrow[R \mid Q]{\alpha}_{H} P' \mid Q} \; (Par_{1,2})$$

Evaluating Blocking Conditions

$$\frac{P \xrightarrow[R_1]{\ell}_{H_1} P' \quad Q \xrightarrow[R_2]{\bar{\ell}}_{H_2} Q' \quad \begin{array}{r} H = \{\tau \mid H_2 \cap \overline{\mathrm{iA}}(P) \nsubseteq \{\bar{\ell}\} \\ \text{or } H_1 \cap \overline{\mathrm{iA}}(Q) \nsubseteq \{\ell\}\} \end{array}}{P \mid Q \xrightarrow[R_1 \mid R_2]{\ell \mid \bar{\ell}}_{H_1 \cup H_2 \cup H} P' \mid Q'} \; (Par_3)$$

Initial Actions: $\mathrm{iA}(P) \stackrel{def}{=} \{\ell \mid \exists H, R, P'.\, P \xrightarrow[R]{\ell}_{H} P'\} \subseteq \mathcal{L}$

Weak Enabling & Phillips' CCS$^{\mathsf{Ph}}$

Weakly Enabled Transitions

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

CCS$^{\mathsf{Ph}}$ Processes

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

$$P \xrightarrow[R]{\alpha}_H P' \text{ is weakly enabled}$$

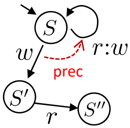
iff $P \xrightarrow{\alpha}_H P'$ is derivable in the semantics of CCS$^{\mathsf{Ph}}$ [Phillips 2001].

Examples - Write-before-Read

Write-before-read Store:

$S = w.S' + r{:}w.S$ and $S' = r.S''$

Concurrent Environment: $E = \bar{r} \mid \overline{w}$



- The transition ("read first")

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

  is not weakly enabled, since $\{w\} \cap \overline{\mathrm{iA}}(0 \mid 0 \mid \overline{w}) = \{w\} \neq \{\,\}$.

- Problem: Weak enabling does not eliminate data races, instead we need...
- The transition ("write first")

$$S \mid E \equiv (w.S' + r{:}w.S) \mid \bar{r} \mid \overline{w} \xrightarrow[0 \mid \bar{r} \mid 0]{w \mid \overline{w}}_{\{\,\}} S' \mid \bar{r} \mid 0$$

  is weakly enabled, since $\{\,\} \cap (\overline{\mathrm{iA}}(0 \mid \bar{r} \mid 0) \cup \{\tau\}) = \{\,\}$.

Twistit I

Constructive Enabling

Constructive Enabling

Constructively Enabled Transitions

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

Potential Actions

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

Note:

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

* α 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 $\alpha_1 \notin H_2$ and $\alpha_2 \notin H_1$.

Coherence

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

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

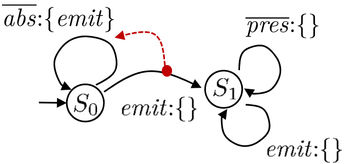
where the c-actions $\alpha_i{:}H_i[E_i]$ are independent or $\alpha_1 = \alpha_2$ and $Q_1 \not\equiv Q_2$. Then, there exist $Q_1' \equiv Q_2'$ and c-enabled transitions

$$Q_1 \xrightarrow[E_2']{\alpha_2}_{H_2'} Q_1' \text{ and } Q_2 \xrightarrow[E_1']{\alpha_1}_{H_1'} Q_2'.$$

Coherent Sharing and Memory

The following are not confluent in CCS but coherent in Synpa$^{\text{tick}}$:

Esterel Signal (pure temporary, no clock):



$$S_0 \overset{df}{=} \overline{abs}{:}emit.S_0 + emit.S_1$$
$$S_1 \overset{df}{=} \overline{pres}.S_1 + emit.S_1$$

→ permits multiple programs on co-names $\overline{emit}$, abs, $pres$.

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

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

→ assumes there is a single signal on co-names $emit$, $\overline{abs}$, $\overline{pres}$.

Twistit III
Policies & Preservation of Coherence

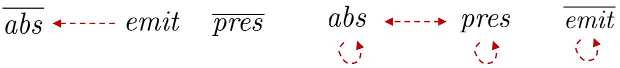
Precedence Policy

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

Precedence Policy

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

Policy Type π_{sig} of Esterel Signals and Programs



Pivot Policy

The p-policy π_{sig} has a special property...

Pivot Policy

A p-policy $\pi = (L, \dashrightarrow)$ 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 π, then $P \mid Q$ is coherent* and conforms to π.

*Since we do not need to restrict we permit sharing!

Conclusion

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!