Universität Bamberg

Coherence and Determinacy with Priorities and Clocks

Claude Stolze

joint work with Michael Mendler and Luigi Liquori

Otto-Friedrich Universität Bamberg
Centre INRIA d'Université Côte d'Azur

Motivations and context

- Aim: present the semantics of Synchronous Programming (SP) in the framework of Process Algebras
  - Make SP more appealing to other communities interested in concurrent programming and concurrency theory
  - Develop and study a shallow encoding of (multiclock) SP languages in process algebras
- Our basic framework: CCS with clocks and priorities (CCS^{spt})
  - Previous (async.) systems with clocks [Hoare]
  - Previous (async.) systems with priorities [Camilleri & Winskel, Phillips]
- CCS^{spt} combines clocks and priorities in a synchronous setting

What is CCS^{spt}?

- CCS^{spt} is a process calculus with
  - CCS (rendezvous) actions for communication [Milner]
  - CSP (broadcast) actions for clocks [Hoare]
  - Priorities for constructive scheduling [Camilleri & Winskel, Phillips]
  - NEW: multiclock (GALS, . . .)
  - NEW: both reduction semantics and labelled transition semantics
- A first glance of CCS^{spt} was presented in Synchron 2023 (Kiel), an encoding single-clock Esterel was presented in Synchron 2024 (Bamberg)

Syntax

$$
\begin{array}{lcll}
P, Q & ::= & M & \text{thread} \\
 & | & \mathrm{p} & \text{process name, } \mathrm{p} \in \mathcal{I} \\
 & | & P \mid Q & \text{parallel composition} \\
 & | & P \setminus A & \text{restriction, } A \subseteq \mathcal{A}. \\
 & | & P \,/\, C & \text{hiding, } C \subseteq \mathcal{C} \\
M, N & ::= & 0_C & \text{inactive process} \\
 & | & \alpha{:}L.P & \text{prefix, } L \subseteq \mathcal{L} \\
 & | & M + N & \text{sum}
\end{array}
$$

Example:

$$
\begin{array}{lcl}
\mathit{User} & \stackrel{\mathit{def}}{=} & \sigma.\sigma.\overline{\mathit{ctrlc}}.0_\sigma \\
\mathit{Prog} & \stackrel{\mathit{def}}{=} & \overline{\mathit{hello}}{:}\mathit{ctrlc}.\sigma.\mathit{Prog} + \mathit{ctrlc}.0_\sigma \\
\mathit{Main} & \stackrel{\mathit{def}}{=} & (\mathit{User} \mid \mathit{Prog}) \,/\, \sigma \setminus \mathit{ctrlc}
\end{array}
$$

Process names have arguments

Dealing with multiple clocks

- clocks(P) is the set of clocks which are active for (read "free in") P
- P has to synchronize on clocks(P), but not on other clocks
- clocks(0_C) $= C$: it blocks clocks only in set C, and does not care about other clocks
- Example: $0_\sigma \mid \rho.P$ can perform $\rho.P$, but $0_\rho \mid \rho.P$ is stuck
- With multiclocks:
  - $\sigma.0_{\{\sigma,\rho'\}} \mid \rho.0_\rho \mid \rho'.0_{\rho'}$ can perform either σ or ρ, but not ρ'
- A process P is well-defined (noted WD(P)) if clocks(P) is constant throughout its execution
- Example: $a.\sigma.0_\sigma + b.0_\sigma$ is well-defined
- Counter-example: $a.\rho.0_\sigma$ is not well-defined. Indeed, if it performs a and then ρ, it loses the clock ρ
- Another counter-example: $a.0_\sigma + b.0_\rho$

Scheduling with multiple clocks

- Each process has to be scheduled according to its clocks and the blocking sets of its actions
- There are 2 issues to consider
- Issue 1: $a{:}b.0_\sigma \mid \rho.\overline{b}.0_\rho$ cannot perform a. Indeed, it is in concurrence with $\rho.\overline{b}.0_\rho$ which can perform $\overline{b}$ inside a σ-cycle
  - In order to schedule the actions of P in a process $P \mid Q$, we have to consider the actions of Q up to clocks(P). This set is called $\mathrm{iA}^*_{\mathrm{clocks}(P)}(Q)$
- Issue 2: Scheduling a non deterministic choice $+$ has to consider immediate conflicts.

  Ex: $a{:}b.0_\sigma \mid (c.\overline{b}.0_\sigma + \overline{a}.0_\sigma)$ cannot perform a, but the τ-transition would make $\overline{b}$ disappear, so we can do it.

  However: $a{:}b.0_\sigma \mid (\overline{b}.0_\sigma + \overline{a}.0_\sigma)$ should not be able to perform either a or τ (immediate preemption)
  - In that case, we consider the simple initial actions iA(P), i.e. the actions a process can do immediately

Labelled Transition System

Example:

$$\begin{aligned} \mathit{User} &\stackrel{\mathit{def}}{=} \sigma.\sigma.\overline{\mathit{ctrlc}}.0_\sigma \\ \mathit{Prog} &\stackrel{\mathit{def}}{=} \overline{\mathit{hello}}:\mathit{ctrlc}.\sigma.\mathit{Prog} + \mathit{ctrlc}.0_\sigma \\ \mathit{Main} &\stackrel{\mathit{def}}{=} (\mathit{User} \mid \mathit{Prog}) \,/\, \sigma \setminus \mathit{ctrlc} \end{aligned}$$

- LTS: $P \xrightarrow[B]{\alpha}_\iota Q$ means P can become Q with action α and the blocking constraints B, and the prediction information ι (omitted)

$$\begin{aligned} \mathit{Prog} &\xrightarrow[(\sigma,\mathit{ctrlc})]{\overline{\mathit{hello}}} \sigma.\mathit{Prog} \\ \mathit{User} \mid \sigma.\mathit{Prog} &\xrightarrow[(\sigma,\{\})]{\sigma} \sigma.\overline{\mathit{ctrlc}}.0_\sigma \mid \mathit{Prog} \end{aligned}$$

- Blocking constraints: each action has a set of contributing threads, each one with its set of clocks and a set of blocking actions
  Ex:
  $\sigma{:}\{a,b\}.0_\sigma \mid \sigma{:}b.0_{\sigma,\rho} \mid \sigma{:}c.0_{\sigma,\rho'} \xrightarrow[(\sigma,\{a,b\}),(\{\sigma,\rho\},b),(\{\sigma,\rho'\},c)]{\sigma} 0_\sigma \mid 0_{\sigma,\rho} \mid 0_{\sigma,\rho'}$

Term Rewriting System

- The Term Rewriting System explains how processes reduce internally:

$$hello.hello \mid Main \longrightarrow hello \mid (User \mid \sigma.Prog) \,/\, \sigma \setminus ctrlc$$

- We can reduce only if we know everything about the blocking actions, e.g.

$$a{:}b \mid \overline{a}$$

cannot reduce, because I don't know if we can do b, but

$$(a{:}b \mid \overline{a}) \setminus b$$

can reduce

Equivalence

$$
\begin{aligned}
M_1 + (M_2 + M_3) &\equiv (M_1 + M_2) + M_3 \\
M_1 + M_2 &\equiv M_2 + M_1 \\
M + 0_{\mathrm{clocks}(M)} &\equiv M \\
\\
P_1 \mid (P_2 \mid P_3) &\equiv (P_1 \mid P_2) \mid P_3 \\
P_1 \mid P_2 &\equiv P_2 \mid P_1 \\
P \mid 0_{\{\}} &\equiv P \\
\\
P \setminus A &\equiv P && \text{if } \mathcal{L}(P) \cap (A \cup \overline{A}) = \{\} \\
P \setminus A_1 \setminus A_2 &\equiv P \setminus (A_1 \cup A_2) \\
P \mid (Q \setminus A) &\equiv (P \mid Q) \setminus A && \text{if } \mathcal{L}(P) \cap (A \cup \overline{A}) = \{\} \\
\\
0_{C_1} \,/\, C_2 &\equiv 0_{C_1 - C_2} \\
P \,/\, C &\equiv P && \text{if } \mathcal{L}(P) \cap C = \{\} \\
P \,/\, C_1 \,/\, C_2 &\equiv P \,/\, (C_1 \cup C_2) \\
P \setminus A \,/\, C &\equiv P \,/\, C \setminus A \\
P \mid (Q \,/\, C) &\equiv (P \mid Q) \,/\, C && \text{if } \mathcal{L}(P) \cap C = \{\} \\
\\
\mathrm{p} &\equiv P && \text{if } \mathrm{p} \stackrel{def}{=} P
\end{aligned}
$$

Reduction semantics

$$\textit{Definition} : \mathsf{allows}(\iota, C, L) \stackrel{def}{=} \iota(C) \cap \overline{L} = \{\}$$

$$\frac{\mathsf{allows}(\mathsf{iA}^*\_(R), \mathsf{clocks}(P), L) \quad L \subseteq C \cup A \cup \overline{A}}{((\tau{:}L.P + M) \mid R) \,/\, C \setminus A \longrightarrow (P \mid R) \,/\, C \setminus A} \; (\textit{R-Tau})$$

$$\frac{\begin{array}{c} L \cup L' \subseteq C \cup A \cup \overline{A} \\ \mathsf{allows}(\mathsf{iA}^*\_(R) \cup (\mathsf{iA}(N) - \{\overline{a}\}), \mathsf{clocks}(P), L) \\ \mathsf{allows}(\mathsf{iA}^*\_(R) \cup (\mathsf{iA}(M) - \{a\}), \mathsf{clocks}(Q), L') \end{array}}{((a{:}L.P + M) \mid (\overline{a}{:}L'.Q + N) \mid R) \,/\, C \setminus A \longrightarrow (P \mid Q \mid R) \,/\, C \setminus A} \; (\textit{R-Com})$$

$$\frac{\begin{array}{c} \bigcup_i L_i \subseteq C \cup A \cup \overline{A} \qquad \sigma \in C \qquad \sigma \notin \mathsf{clocks}(R) \\ \forall i, \mathsf{allows}(\mathsf{iA}^*\_(R) \cup (\bigcup_{j \neq i} \mathsf{iA}(M_j) - \{\sigma\}), \mathsf{clocks}(P_i), L_i) \end{array}}{(\sigma{:}L_1.P_1 + M_1) \mid \cdots \mid (\sigma{:}L_n.P_n + M_n) \mid R) \,/\, C \setminus A \longrightarrow (P_1 \mid \cdots \mid P_n \mid R) \,/\, C \setminus A} \; (\textit{R-Clk})$$

Labelled Transition System

$$\textit{Definition} : \text{allows}(\iota, B) \stackrel{def}{=} \forall (C, L) \in B, \iota(C) \cap \overline{L} = \{\}$$

$$\frac{}{\alpha{:}L.P \xrightarrow[\{(\text{clocks}(P),L)\}]{\alpha} \iota\{\}\ P} \ (Act)$$

$$\frac{M_1 \xrightarrow[B]{\alpha} \iota\ P}{M_1 + M_2 \xrightarrow[B]{\alpha} \iota \cup (\text{iA}(M_2) - \{\alpha\})\ P} \ (Sum_1)$$

$$\frac{P_1 \xrightarrow[B_1]{\ell} \iota_1\ P_1' \quad P_2 \xrightarrow[B_2]{\overline{\ell}} \iota_2\ P_2' \quad \text{allows}(\iota_1, B_2) \quad \text{allows}(\iota_2, B_1)}{P_1 \mid P_2 \xrightarrow[B_1 \cup B_2]{\ell \mid \overline{\ell}} \iota_1 + \iota_2\ P_1' \mid P_2'} \ (Com)$$

$$\frac{P \xrightarrow[B]{\alpha} \iota\ P' \quad \text{allows}(\text{iA}^*_{\_}(Q), B) \quad \alpha \notin \text{clocks}(Q)}{P \mid Q \xrightarrow[B]{\alpha} \iota + \text{iA}^*_{\_}(Q)\ P' \mid Q} \ (Par_1)$$

Restriction acts as in CCS, and hiding acts as in CSP

Properties

- The Term Rewriting System and the Labelled Transition System are in harmony, i.e. their semantics coincide

Lemma (Harmony)

$$P \longrightarrow Q \text{ iff } P \xrightarrow[B]{\tau}_{\iota} Q' \equiv Q$$

for some ι, Q', B *such that* $\forall (C, L) \in B, L = \{\}$

Examples with multiple clocks

- Looping on clocks in C:

$$\mathrm{halt}_C \stackrel{def}{=} \Sigma_{\sigma \in C} \sigma.\mathrm{halt}_C$$

- Get a string from keystrokes (at most one keystroke per ρ-cycle):

$$\begin{aligned} \mathrm{GetString} \stackrel{def}{=} & \ \mathit{return}.\overline{\mathit{string}}.\mathrm{halt}_\rho \\ + & \ \mathit{key}{:}\mathit{return}.\rho.\mathrm{GetString} \\ + & \ \rho{:}\{\mathit{return}, \mathit{key}\}.\mathrm{GetString} \end{aligned}$$

- Getting a string in a σ-cycle:

$$\mathrm{GetString} \mid (\mathit{string}.\sigma.Q_1 + \sigma{:}\mathit{string}.Q_2)$$

- This is deterministic, thanks to priorities
- As GetString does not do σ-transition, $\mathrm{iA}^*_{\{\sigma\}}(\mathrm{GetString})$ contains *string*, so it blocks the σ transition
- ρ is temporarily a “refinement” of σ [cf. Gemünde]

Examples with multiple clocks

- GetString with a bit of asynchrony:

$$\begin{aligned} \text{GetString} \stackrel{def}{=} & \; \mathit{return}.(\overline{\mathit{string}}.0_{\{\}} \mid \text{halt}_\rho) \\ + & \; \mathit{key}{:}\mathit{return}.\rho.\text{GetString} \\ + & \; \rho{:}\{\mathit{return}, \mathit{key}\}.\text{GetString} \end{aligned}$$

- Getting a string asynchronously (in the spirit of GALS):

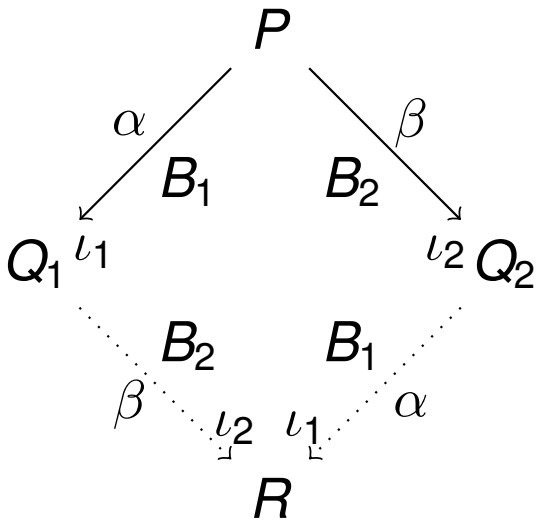
$$\text{Bridge} \stackrel{def}{=} \mathit{string}.\overline{\mathit{ok}}.\text{halt}_\sigma + \sigma.\text{Bridge}$$

- Bridge sends the $\overline{ok}$ signal in the same cycle it received the string
- Bridge does not follow the maximal clock progress principle, it is not deterministic either
- $P \stackrel{def}{=} \mathit{ok}.\sigma.P' + \sigma{:}\mathit{ok}.P$
- We can consider

$$\text{GetString} \mid \text{Bridge} \mid P$$

Conclusion and Future Work

- Define a reasonable Church-Rosser property (Coherence)
- a property studied by Milner in Communication and Concurrency



- The top (rendez-vous) transitions in the CR diagram should not be mutually exclusive
- Formally, $\forall (C, L) \in B_2, \alpha \notin L$ and $\forall (C, L) \in B_1, \beta \notin L$
- Ex: in $a{:}b + b$, the a- and b-transition are mutually exclusive
- Policies could help us ensure coherence
- Implementing a software prototype (WIP)