Deadlock Freedom for Asynchronous and Cyclic Process Networks*

Bas van den Heuvel and Jorge A. Pérez

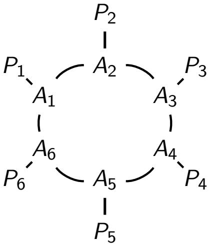
Bernoulli Institute for Math, CS, and AI
University of Groningen, The Netherlands

ICE 2021

*Research partially supported by the Dutch Research Council (NWO) under project No. 016.Vidi.189.046 (Unifying Correctness for Communicating Software).

Introduction

- Goal: study the fundamentals of deadlock freedom for cyclic process networks with asynchronous communication.
  - What is a cyclic process network?
  - Why is deadlock freedom hard?
  - Why asynchronous communication?



- Setting: behavioral type systems derived from Linear Logic through Curry-Howard.
- Our contribution: APCP, session type system for deadlock free π-calculus processes with *asynchronous* communication that supports *cyclic process networks* and *tail-recursion*.

  (based on Dardha & Gay, and on DeYoung et al.)
- Presentation outline: introduce syntax, semantics and typing, discuss Milner's cyclic scheduler, conclude.

Process Syntax and Semantics

Syntax:

$x(y,z);P$	input	|	$(\boldsymbol{\nu}xy)P$	restriction
$x[y,z]$	output	|	$P \mid Q$	parallel
$x(z) \triangleright \{i : P_i\}_{i \in I}$	branching	|	$\mathbf{0}$	inaction
$x[z] \triangleleft j$	selection	|	$x \leftrightarrow y$	forwarder
$\mu X(\tilde{x});P$	recursive definition	|	$X\langle\tilde{x}\rangle$	recursive call

Semantics:

$$(\boldsymbol{\nu}xy)(x[a,b] \mid y(v,z);P) \longrightarrow P\{a/v,b/z\}$$

$$(\boldsymbol{\nu}xy)(x[b] \triangleleft j \mid y(z) \triangleright \{i : P_i\}_{i \in I}) \longrightarrow P_j\{b/z\}$$

$$(\boldsymbol{\nu}xy)(x \leftrightarrow z \mid P) \longrightarrow P\{z/y\}$$

Process Syntax and Semantics: Binding Continuations

Outputs $x[a, b]$ have no continuation, but can be bound to one using parallel and restriction. *Syntactic sugar* is useful:

$$\overline{x}[u] \cdot P := (\boldsymbol{\nu} au)(\boldsymbol{\nu} bx')(x[a, b] \mid P\{x'/x\})$$

Similarly for selection:

$$\overline{x} \triangleleft j \cdot P := (\boldsymbol{\nu} bx')(x[b] \triangleleft j \mid P\{x'/x\})$$

Typing

Types:

$A \parr^{o} B$	input	$\mid A \otimes^{o} B$	output
$\&^{o}\{i : A_i\}_{i \in I}$	branching	$\mid \oplus^{o}\{i : A_i\}_{i \in I}$	selection
		$\mid \bullet$	closed session
$\mu X.A$	recursive definition	$\mid X$	recursive call

$$A, B ::= A \otimes^{o} B \mid A \parr^{o} B \mid \oplus^{o}\{i : A_i\}_{i \in I} \mid \&^{o}\{i : A_i\}_{i \in I} \mid \bullet \mid \mu X.A \mid X$$

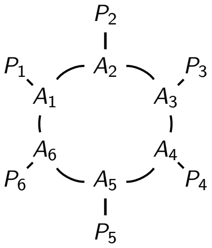
$$\overline{A}, \overline{B} ::= \overline{A} \parr^{o} \overline{B} \mid \overline{A} \otimes^{o} \overline{B} \mid \&^{o}\{i : \overline{A_i}\}_{i \in I} \mid \oplus^{o}\{i : \overline{A_i}\}_{i \in I} \mid \bullet \mid \mu X.\overline{A} \mid X$$

Typing:

$$\frac{P \vdash \Gamma, y{:}A, z{:}B \qquad \mathsf{o} < \mathrm{pr}(\Gamma)}{x(y,z); P \vdash \Gamma, x{:}A \parr^{o} B} \parr \qquad \frac{\text{no priority checks}}{x[y,z] \vdash x{:}A \otimes^{o} B, y{:}\overline{A}, z{:}\overline{B}} \otimes$$

$$\frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta} \textsc{Mix} \qquad \frac{P \vdash \Gamma, x{:}A, y{:}\overline{A}}{(\boldsymbol{\nu} xy)P \vdash \Gamma} \textsc{Cycle}$$

Milner's Cyclic Scheduler



Each scheduler A_i has three endpoints:

- a_i to connect with P_i on b_i,
- c_{i-1} to connect with A_{i-1}, and
- d_i to connect with A_{i+1}.

Complete scheduler with n workers:

$Sched_n := (\boldsymbol{\nu} c_i d_i)_{1 \leq i \leq n}(\prod_{1 \leq i \leq n}(\boldsymbol{\nu} a_i b_i)(A_i \mid P_i))$

$$\begin{aligned} A_{i+1} := \mu X(a_{i+1}, c_i, d_{i+1}); & \, c_i \triangleright \text{start}; a_{i+1} \triangleleft \text{start} \cdot d_{i+1} \triangleleft \text{start} \cdot a_{i+1} \triangleright \text{ack}; \\ & \, c_i \triangleright \text{next}; d_{i+1} \triangleleft \text{next} \cdot X\langle a_{i+1}, c_i, d_{i+1} \rangle \end{aligned}$$

$$\begin{aligned} A_1 := \mu X(a_1, c_n, d_1); & \, d_1 \triangleleft \text{start} \cdot a_1 \triangleleft \text{start} \cdot a_1 \triangleright \text{ack}; \\ & \, d_1 \triangleleft \text{next} \cdot c_n \triangleright \text{start}; c_n \triangleright \text{next}; X\langle a_1, c_n, d_1 \rangle \end{aligned}$$

$Sched_n \vdash \emptyset$ so **deadlock free**

Conclusion

- APCP: type system for deadlock freedom of cyclic process networks with asynchronous communication and recursion.
- Main take-away: asynchrony significantly simplifies priority management compared to synchronous PCP.
- Ongoing work: use APCP to typecheck process implementations of participants in multiparty protocols.