Logical Foundations for Concurrency: Non-Determinism and Failures

Jorge A. Pérez
University of Groningen, The Netherlands
j.a.perez [[at]] rug.nl

ECI 2026
(Version of 31st July 2026)

Preamble

- This presentation is based on joint work with Luís Caires (Lisbon).
- The paper was originally presented at ESOP 2017 (European Symposium on Programming Language and Systems).
- Although it has been a while, over the years the ideas in this work have proven influential in several ways. In particular:
  - New encodings of λ-calculi with **intersection types** as session-typed processes (jww Daniele Nantes and Joe Paulus)
  - New process languages with **shared state**, later implemented as the **CLASS programming language** (Pedro Rocha and Luís Caires)
- Many exciting things to do!
  (See suggested readings for topics we couldn't cover).

Outline

Context

Logical Foundations for Concurrency
- Curry-Howard correspondence for Concurrency
- Example: Movie Server

Adding Non Determinism and Failure
- Syntax and Semantics
- Types
- Properties

Examples
- Extended Movie Server
- Encoding λ^{exc}

Concluding Remarks

Behavioral Types (1/2)

By classifying **values**, traditional data types are an effective basis for validating sequential programs

To reason about services, **behavioral types** classify **interactions**

- High-level representations of communication structures
- Compositional tools for (statically) checking service behavior
- Tied to programming abstractions that promote communication as a first-class concern

Behavioral Types (2/2)

- Typically developed upon **process calculi**
  - ⋆ Variants of the π-calculus
  - ⋆ Expressive core programming models; suited for investigation
- Formal specification languages, based on communication
  - ⋆ Strong ties with automata, logic, type theory
  - ⋆ Clear linkage with validation methods
  - ⋆ Precise notions of runtime correctness
- General verification techniques tailored to actual languages (Java, Scala, ML, Haskell). Many recent advances.
- A notable class: **session types**

Session Types (1/2)

Seminal type-based approach to the analysis of structured communications [Honda, Vasconcelos, Kubo (1998)]

- Communication protocols structured into **sessions**
- Concurrent processes interacting along **session channels**
- Disciplined interactive behavior, abstracted as **session types**

Session Types (2/2)

Session specifications usually given as π-calculus processes

- Actions always occur in **dual** pairs
- New sessions created by invoking **shared servers**
- **Concurrency** in the simultaneous execution of sessions
- **Mobility** in the exchange of session and server names

Correctness Guarantees for Specifications

- Adhere to their ascribed session protocols - **Fidelity**
- Do not feature runtime errors – **Safety**
- Do not get stuck – **Progress / Lock-Freedom**
- Do not have infinite reduction sequences – **Termination**

Session Types (2/2)

Session specifications usually given as π-calculus processes

- Actions always occur in **dual** pairs
- New sessions created by invoking **shared servers**
- **Concurrency** in the simultaneous execution of sessions
- **Mobility** in the exchange of session and server names

Correctness Guarantees for Specifications

- Adhere to their ascribed session protocols - **Fidelity**
- Do not feature runtime errors – **Safety**
- Do not get stuck – **Progress / Lock-Freedom**
- Do not have infinite reduction sequences – **Termination**

In this talk: **binary** session types (two-party protocols).

Outline

Context

Logical Foundations for Concurrency
- Curry-Howard correspondence for Concurrency
- Example: Movie Server

Adding Non Determinism and Failure
- Syntax and Semantics
- Types
- Properties

Examples
- Extended Movie Server
- Encoding λ^{exc}

Concluding Remarks

Logical Foundations for Session Types

Curry-Howard correspondence for Concurrency:

propositions	$\leftrightarrow$	session types
sequent proofs	$\leftrightarrow$	π-calculus processes
cut elimination	$\leftrightarrow$	process communication

Developed by Caires & Pfenning for intuitionistic linear logic (2010).
Adapted to classical linear logic by Wadler (2012).

Logical Foundations for Session Types

Curry-Howard correspondence for Concurrency:

propositions	$\leftrightarrow$	session types
sequent proofs	$\leftrightarrow$	π-calculus processes
cut elimination	$\leftrightarrow$	process communication

Developed by Caires & Pfenning for intuitionistic linear logic (2010).
Adapted to classical linear logic by Wadler (2012).

Main Features

- Clear account of resource usage policies in concurrency
- Session fidelity, runtime safety, global progress "for free"
- Excellent basis for generalizations and extensions

Process Model: Syntax

$$
\begin{array}{rll}
P, Q ::= & \overline{x}(y).P \mid x(y).P \mid !x(y).P & \text{[output and (replicated) input]} \\
\mid & P \mid Q \mid (\nu y)P \mid 0 & \text{[parallel, restriction, inaction]} \\
\mid & x.\mathtt{case}(P, Q) & \text{[branching]} \\
\mid & x.\mathtt{inr}; P \mid x.\mathtt{inl}; P & \text{[right and left selection]} \\
\mid & x.\overline{\mathtt{close}} \mid x.\mathtt{close}; P & \text{[session termination]} \\
\mid & [x \leftrightarrow y] & \text{[channel forwarder]}
\end{array}
$$

Process Model: Reduction Semantics [Excerpt]

Reduction ($\longrightarrow$) plus structural congruence ($\equiv$):

$$
\begin{array}{rcl}
\overline{x}(y).Q \mid x(y).P & \longrightarrow & (\nu y)(Q \mid P) \\
\overline{x}(y).Q \mid !x(y).P & \longrightarrow & (\nu y)(Q \mid P) \mid !x(y).P \\
x.\mathtt{inl}; P \mid x.\mathtt{case}(Q, R) & \longrightarrow & P \mid Q \\
x.\overline{\mathtt{close}} \mid x.\mathtt{close}; P & \longrightarrow & P \\
(\nu x)([x \leftrightarrow y] \mid P) & \longrightarrow & P\{y/x\} \\
P \equiv P',\ P' \longrightarrow Q',\ Q' \equiv Q & \Rightarrow & P \longrightarrow Q
\end{array}
$$

Types and Duality (CLL)

Types are assigned to names and describe their session behavior:

$x : A \otimes B$	[Output name of type A, continue as B]
$x : A \parr B$	[Input name of type A, continue as B]
$x : !A$	[Persistent offer of A]
$x : ?A$	[Request of a persistent behavior A]
$x : A \mathbin{\&} B$	[Offer *both* A and B]
$x : A \oplus B$	[Select *either* A or B]
$x : 1$	[Terminated interaction]
$x : \bot$	[Terminated interaction]

Duality of A, written $\overline{A}$, is standard.

Type Judgments

$$P \vdash \underbrace{x_1{:}A_1, \ldots, x_k{:}A_k}_{\Delta \,:\, \text{linear services}} \;;\; \underbrace{u_1{:}B_1, \ldots, u_n{:}B_n}_{\Theta \,:\, \text{shared services}}$$

- Judgments correspond to logical sequents in the classical linear logic Σ_2 [Andreoli'92]
- Supports exponentials $!A$ and $?A$ in terms of standard semantics for lazy replication in the π-calculus

Sample Typing Rules: Identity, Mix & Cut

$$\frac{}{\vdash A, \overline{A}; \Theta}\,(\mathsf{Tid})$$

$$\frac{\vdash \Delta; \Theta \qquad \vdash \Delta'; \Theta}{\vdash \Delta, \Delta'; \Theta}\,(\mathsf{T}\mid)$$

$$\frac{\vdash \Delta, A; \Theta \qquad \vdash \Delta', \overline{A}; \Theta}{\vdash \Delta, \Delta'; \Theta}\,(\mathsf{Tcut})$$

Sample Typing Rules: Identity, Mix & Cut

$$\frac{}{[x \leftrightarrow y] \vdash x{:}A,\, y{:}\overline{A};\, \Theta}\,(\mathsf{Tid})$$

$$\frac{P \vdash \Delta;\, \Theta \qquad Q \vdash \Delta';\, \Theta}{P \mid Q \vdash \Delta, \Delta';\, \Theta}\,(\mathsf{T}\mid)$$

$$\frac{P \vdash \Delta, x{:}A;\, \Theta \qquad Q \vdash \Delta', x{:}\overline{A};\, \Theta}{(\nu x)(P \mid Q) \vdash \Delta, \Delta';\, \Theta}\,(\mathsf{Tcut})$$

Sample Typing Rules: Input & Output

$$\frac{P \vdash \Delta, y{:}A; \Theta \quad Q \vdash \Delta', x{:}B; \Theta}{\overline{x}(y).(P \mid Q) \vdash \Delta, \Delta', x{:}A \otimes B; \Theta}\ (\mathsf{T}\otimes)$$

$$\frac{P \vdash \Delta, y{:}C, x{:}D; \Theta}{x(y).P \vdash \Delta, x{:}C \parr D; \Theta}\ (\mathsf{T}\parr)$$

Sample Typing Rules: Input & Output

$$\frac{P \vdash \Delta, y{:}A; \Theta \quad Q \vdash \Delta', x{:}B; \Theta}{\overline{x}(y).(P \mid Q) \vdash \Delta, \Delta', x{:}A \otimes B; \Theta}\ (\mathsf{T}\otimes)$$

$$\frac{P \vdash \Delta, y{:}C, x{:}D; \Theta}{x(y).P \vdash \Delta, x{:}C \parr D; \Theta}\ (\mathsf{T}\parr)$$

Reduction:

$$(\nu x)(\overline{x}(y).(P \mid Q) \mid x(y).R) \longrightarrow (\nu x)(\nu y)(P \mid Q \mid R)$$

[Free output $x\langle y\rangle.P$ representable as $\overline{x}(z).(P \mid [z \leftrightarrow y])$.]

Sample Typing Rules: Exponentials

$$\frac{\vdash A; \Theta}{\vdash\, !\, A; \Theta}(\mathsf{T}!) \qquad \frac{\vdash \Delta, A; A, \Theta}{\vdash \Delta; A, \Theta}(\mathsf{Tcopy})$$

$$\frac{\vdash \Delta;\ A, \Theta}{\vdash \Delta,\ ?A; \Theta}(\mathsf{T}?) \qquad \frac{\vdash \overline{A}; \Theta \quad \vdash \Delta; A, \Theta}{\vdash \Delta; \Theta}(\mathsf{Tcut}^{?})$$

Sample Typing Rules: Exponentials

$$\frac{P \vdash y{:}A; \Theta}{!x(y).P \vdash x{:}!\,A; \Theta}(\mathsf{T}!) \qquad \frac{P \vdash \Delta, y{:}A; x{:}A, \Theta}{\overline{x}(y).P \vdash \Delta; x{:}A, \Theta}(\mathsf{Tcopy})$$

$$\frac{P \vdash \Delta;\ x{:}A, \Theta}{P \vdash \Delta, x{:}?A; \Theta}(\mathsf{T}?) \qquad \frac{P \vdash y{:}\overline{A}; \Theta \quad Q \vdash \Delta; x{:}A, \Theta}{(\nu x)(!x(y).P \mid Q) \vdash \Delta; \Theta}(\mathsf{Tcut}^{?})$$

Reduction:

$$(\nu x)(!x(y).P \mid \overline{x}(y).R) \longrightarrow (\nu x)(!x(y).P \mid (\nu y)(R \mid P))$$

Main Results

Below $\cong_s$ denotes a process congruence capturing proof conversions (computational, structural, commuting):

Theorem (Cut Elimination)

If $P \vdash \Delta; \Theta$ *then there is a process* Q *such that* $P \cong_s Q$ *and* $Q \vdash \Delta; \Theta$ *is derivable without using rules* $(T\mathrm{cut})$ *and* $(T\mathrm{cut}^?)$.

Theorem (Type Preservation)

If $P \vdash \Delta; \Theta$ *and* $P \to Q$ *then* $Q \vdash \Delta; \Theta$.

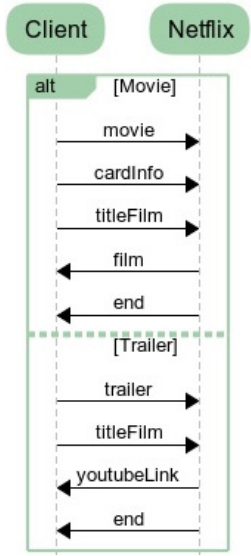
Write $\mathit{live}(P)$ whenever a prefix (non replicated) is at top-level:

Theorem (Progress)

If $P \vdash ; \Theta$ *and* $\mathit{live}(P)$ *then there is* Q *such that* $P \to Q$.

Also: strong normalization and confluence hold [Pérez et al.'12].

Example: A Simple Movie Server



Example: A Simple Movie Server

$$SBody(s) \triangleq s.\mathtt{case}(s(title).s(card).\overline{s}(movie).s.\overline{\mathtt{close}},\ s(title).\overline{s}(trailer).s.\overline{\mathtt{close}}\,)$$

$$Alice(s) \triangleq s.\mathtt{inr};\overline{s}(\text{“}Odyssey\text{”}).s(preview).s.\mathtt{close};0$$

$$Bob(s) \triangleq s.\mathtt{inl};\overline{s}(\text{“}Matrix\text{”}).\overline{s}(bobscard).s(mpeg).s.\mathtt{close};0$$

$$System_1 \triangleq (\nu s)(SBody(s) \mid Alice(s))$$

Example: A Simple Movie Server

$$SBody(s) \triangleq s.\mathtt{case}(s(title).s(card).\overline{s}(movie).s.\overline{\mathtt{close}},\ s(title).\overline{s}(trailer).s.\overline{\mathtt{close}})$$

$$Alice(s) \triangleq s.\mathtt{inr}; \overline{s}(\text{“}Odyssey\text{”}).s(preview).s.\mathtt{close}; 0$$

$$Bob(s) \triangleq s.\mathtt{inl}; \overline{s}(\text{“}Matrix\text{”}).\overline{s}(bobscard).s(mpeg).s.\mathtt{close}; 0$$

$$System_1 \triangleq (\nu s)(SBody(s) \mid Alice(s))$$

Types

Recall that $A \multimap B$ and $\overline{A} \parr B$ are equivalent.

Assuming basic types `Title`, `CCard`, and `Mov` we have:

$$SBT \triangleq (\mathtt{Title} \multimap \mathtt{CCard} \multimap \mathtt{Mov} \otimes 1)\ \&\ (\mathtt{Title} \multimap \mathtt{Mov} \otimes 1)$$

$$SBody(s) \vdash s : SBT\ ; \cdot$$

$$Alice(s) \vdash s : \overline{SBT}\ ; \cdot \qquad Bob(s) \vdash s : \overline{SBT}\ ; \cdot$$

Movie Server, with Infinite Behavior

$$Movies(srv) \triangleq !srv(s).SBody(s)$$

$$Bob(s) \triangleq s.\texttt{inl}; \overline{s}(\text{“}Matrix\text{”}).\overline{s}(bobscard).s(mpeg).s.\texttt{close}; 0$$

$$SAlice(srv) \triangleq \overline{\overline{srv}}(s).Alice(s)$$

$$SBob(srv) \triangleq \overline{\overline{srv}}(s).Bob(s)$$

$$System_2 \triangleq (\nu srv)(Movies(srv) \mid SAlice(srv) \mid SBob(srv))$$

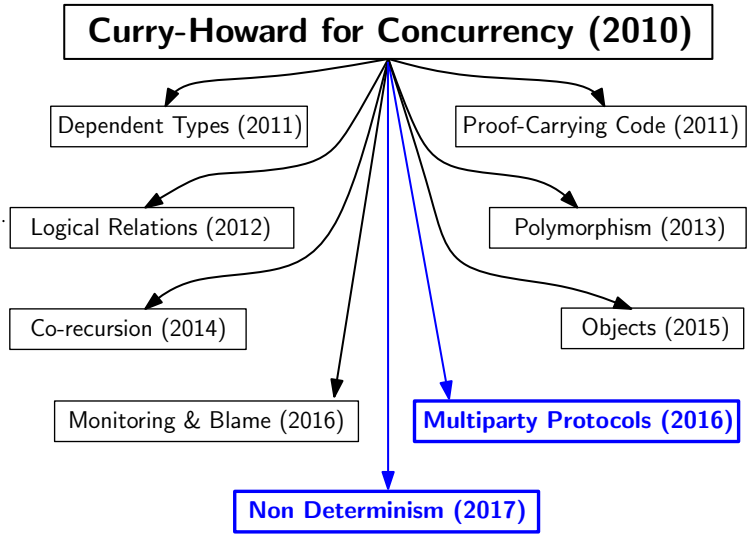
Movie Server, with Infinite Behavior

$$
\begin{aligned}
\mathit{Movies}(\mathit{srv}) &\triangleq !\mathit{srv}(s).\mathit{SBody}(s) \\
\mathit{Bob}(s) &\triangleq s.\mathtt{inl}; \overline{s}(\text{“}\mathit{Matrix}\text{”}).\overline{s}(\mathit{bobscard}).s(\mathit{mpeg}).s.\mathtt{close}; 0 \\
\mathit{SAlice}(\mathit{srv}) &\triangleq \overline{\mathit{srv}}(s).\mathit{Alice}(s) \\
\mathit{SBob}(\mathit{srv}) &\triangleq \overline{\mathit{srv}}(s).\mathit{Bob}(s) \\
\mathit{System}_2 &\triangleq (\nu\, \mathit{srv})(\mathit{Movies}(\mathit{srv}) \mid \mathit{SAlice}(\mathit{srv}) \mid \mathit{SBob}(\mathit{srv}))
\end{aligned}
$$

Types

$$
\begin{aligned}
\mathit{SAlice}(\mathit{srv}) &\vdash \cdot\ ;\ \mathit{srv} : \overline{\mathit{SBT}} \\
\mathit{SBob}(\mathit{srv}) &\vdash \cdot\ ;\ \mathit{srv} : \overline{\mathit{SBT}} \\
\mathit{SAlice}(\mathit{srv}) \mid \mathit{SBob}(\mathit{srv}) &\vdash \mathit{srv} : ?\overline{\mathit{SBT}};\ \cdot \\
\mathit{Movies}(\mathit{srv}) &\vdash \mathit{srv} : !\mathit{SBT}\ ;\ \cdot
\end{aligned}
$$

Many Extensions!



Outline

Context

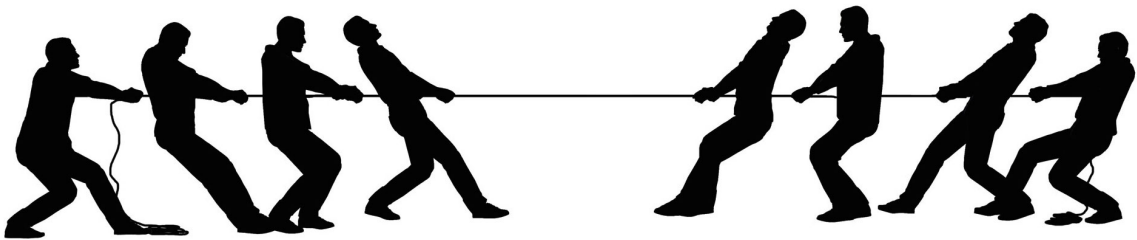
Logical Foundations for Concurrency
- Curry-Howard correspondence for Concurrency
- Example: Movie Server

Adding Non Determinism and Failure
- Syntax and Semantics
- Types
- Properties

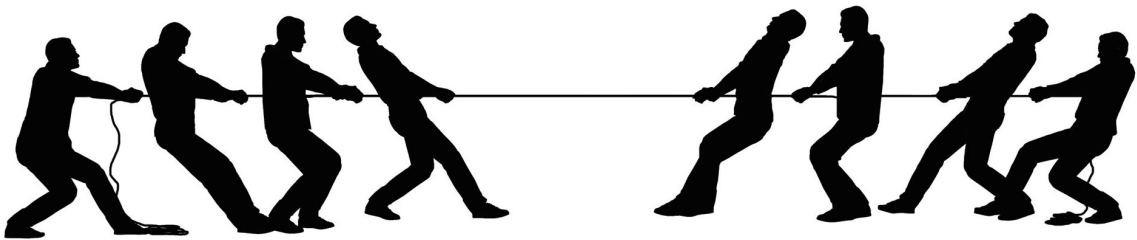
Examples
- Extended Movie Server
- Encoding λ^{exc}

Concluding Remarks

Dealing with Long Known Tensions

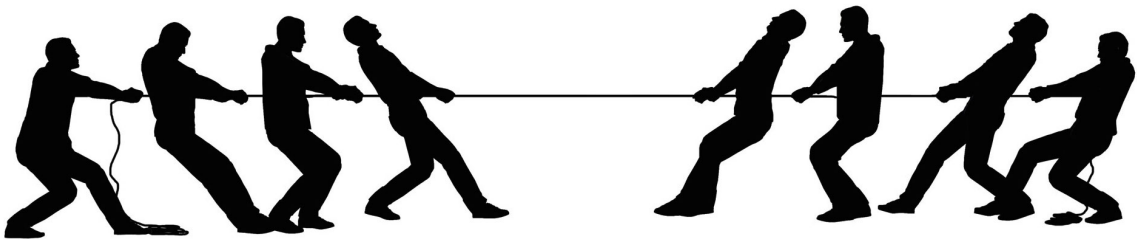


Dealing with Long Known Tensions



Linearity Control effects (e.g. exceptions)

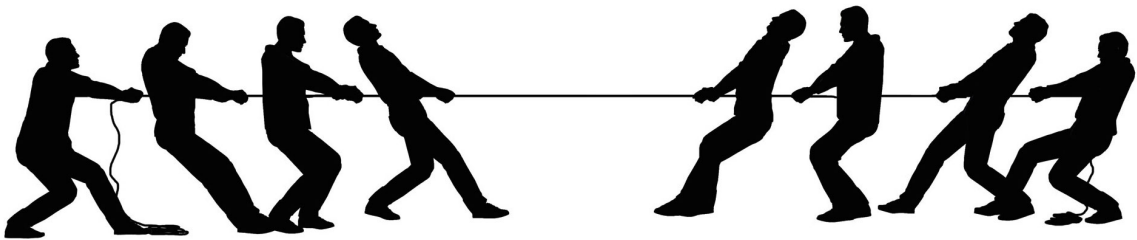
Dealing with Long Known Tensions



Linearity	Control effects (e.g. exceptions)
Propositions as Types	Non determinism

Dealing with Long Known Tensions

Concurrency, distribution, stateful resources, ...



Linearity	Control effects (e.g. exceptions)
Propositions as Types	Non determinism

Context

Much work on solving one or both tensions, including concurrent models and logic-based (Curry-Howard) approaches:

- Classical linear logic and call/cc [Griffin'90]
- Linear continuation passing (e.g. [Berdine et al.'01])
- Session types and interactional exceptions [Carbone et al.'08; Capecchi et al.'10]
- Practical affinity in stateful settings [Tov & Pucella'10,11]
- Affine session types [Mostrous & Vasconcelos'14; Pfenning & Griffith'15]
- Mutually encoding sessions and effects [Orchard & Yoshida'16]

Context

Much work on solving one or both tensions, including concurrent models and logic-based (Curry-Howard) approaches:

- Classical linear logic and call/cc [Griffin'90]
- Linear continuation passing (e.g. [Berdine et al.'01])
- Session types and interactional exceptions [Carbone et al.'08; Capecchi et al.'10]
- Practical affinity in stateful settings [Tov & Pucella'10,11]
- Affine session types [Mostrous & Vasconcelos'14; Pfenning & Griffith'15]
- Mutually encoding sessions and effects [Orchard & Yoshida'16]

Still, no programming model uniformly supports concurrency, linearity, non-determinism, and control effects while

- following logical principles in a Curry-Howard style
- enforcing key properties (progress, SN, confluence)

Our Contributions (1/2)

An extension of the interpretation of classical linear logic as session types, covering **non determinism** and **failure**

Our Contributions (1/2)

An extension of the interpretation of classical linear logic as session types, covering **non determinism** and **failure**

Key Features

- At the **type level**, two new modalities:
  - Type $\& A$ for sessions that **may produce** a behavior of type A
  - Type $\oplus A$ for sessions that **may consume** a behavior of type A
- At the **process level**:
  - Explicit prefixes: $x.\overline{\texttt{some}}; P \quad x.\overline{\texttt{none}}; P \quad x.\texttt{some}_{\overline{w}}; P$
  - Internal non-determinism: $P \oplus Q$

Our Contributions (2/2)

Main results

1. Cut elimination for CLL with new modalities
2. Type preservation, global progress (deadlock-freedom), strong normalization, confluence
3. Compatibility of standard (non confluent) and internal (confluent) non-determinism
4. Type-preserving encodability of λ^{exc}:
   λ-calculus + threads, exceptions, sessions & non-determinism

Process Model: Syntax

$$
\begin{array}{rll}
P, Q ::= & \overline{x}(y).P \mid x(y).P \mid !x(y).P & \text{[output and (replicated) input]} \\
\mid & P \mid Q \mid (\nu y)P \mid 0 & \text{[parallel, restriction, inaction]} \\
\mid & x.\mathtt{case}(P, Q) & \text{[branching]} \\
\mid & x.\mathtt{inr}; P \mid x.\mathtt{inl}; P & \text{[right and left selection]} \\
\mid & x.\overline{\mathtt{close}} \mid x.\mathtt{close}; P & \text{[session termination]} \\
\mid & [x \leftrightarrow y] & \text{[channel forwarder]}
\end{array}
$$

Process Model: Syntax

$$
\begin{array}{rll}
P, Q ::= & \overline{x}(y).P \mid x(y).P \mid !x(y).P & \text{[output and (replicated) input]} \\
\mid & P \mid Q \mid (\nu y)P \mid 0 & \text{[parallel, restriction, inaction]} \\
\mid & x.\mathtt{case}(P, Q) & \text{[branching]} \\
\mid & x.\mathtt{inr}; P \mid x.\mathtt{inl}; P & \text{[right and left selection]} \\
\mid & x.\overline{\mathtt{close}} \mid x.\mathtt{close}; P & \text{[session termination]} \\
\mid & [x \leftrightarrow y] & \text{[channel forwarder]} \\
\\
\mid & x.\overline{\mathtt{some}}; P \quad \mid \quad x.\overline{\mathtt{none}}; P & \textbf{Produce} \text{ non-det behavior} \\
\mid & x.\mathtt{some}_{w_1 \cdots w_n}; P & \textbf{Consume} \text{ non-det behavior} \\
\mid & P \oplus Q & \text{Internal } \textbf{non-determinism}
\end{array}
$$

Process Model: Reduction Semantics [Excerpt]

Reduction ($\longrightarrow$) plus structural congruence ($\equiv$):

$$
\begin{array}{rcl}
\overline{x}(y).Q \mid x(y).P & \longrightarrow & (\nu y)(Q \mid P) \\
\overline{x}(y).Q \mid !x(y).P & \longrightarrow & (\nu y)(Q \mid P) \mid !x(y).P \\
x.\mathtt{inl}; P \mid x.\mathtt{case}(Q, R) & \longrightarrow & P \mid Q \\
x.\overline{\mathtt{close}} \mid x.\mathtt{close}; P & \longrightarrow & P \\
(\nu x)([x \leftrightarrow y] \mid P) & \longrightarrow & P\{y/x\} \\
P \equiv P',\ P' \longrightarrow Q',\ Q' \equiv Q & \Rightarrow & P \longrightarrow Q
\end{array}
$$

Process Model: Reduction Semantics [Excerpt]

Reduction ($\longrightarrow$) plus structural congruence ($\equiv$):

$$
\begin{array}{rcl}
\overline{x}(y).Q \mid x(y).P & \longrightarrow & (\nu y)(Q \mid P) \\
\overline{x}(y).Q \mid !x(y).P & \longrightarrow & (\nu y)(Q \mid P) \mid !x(y).P \\
x.\mathtt{inl}; P \mid x.\mathtt{case}(Q, R) & \longrightarrow & P \mid Q \\
x.\overline{\mathtt{close}} \mid x.\mathtt{close}; P & \longrightarrow & P \\
(\nu x)([x \leftrightarrow y] \mid P) & \longrightarrow & P\{y/x\} \\
P \equiv P',\ P' \longrightarrow Q',\ Q' \equiv Q & \Rightarrow & P \longrightarrow Q
\end{array}
$$

$$
\begin{array}{rcl}
x.\overline{\mathtt{some}}; P \mid x.\mathtt{some}_{w_1 \ldots w_n}; Q & \longrightarrow & P \mid Q \\
x.\overline{\mathtt{none}} \mid x.\mathtt{some}_{w_1 \ldots w_n}; Q & \longrightarrow & w_1.\overline{\mathtt{none}} \mid \ldots \mid w_n.\overline{\mathtt{none}} \\
Q \longrightarrow Q' \Rightarrow P \oplus Q & \longrightarrow & P \oplus Q' \\
P \oplus Q & \equiv & Q \oplus P \\
(\nu x)(P \mid (Q \oplus R)) & \equiv & (\nu x)(P \mid Q) \oplus (\nu x)(P \mid R)
\end{array}
$$

Session Types with Non Determinism

- A Curry-Howard correspondence for concurrency enhancing prior works [Caires & Pfenning'10; Wadler'12]:

$$\begin{array}{rcl} \text{linear logic propositions} & \leftrightarrow & \text{session types} \\ \text{sequent proofs} & \leftrightarrow & \pi\text{-calculus processes} \\ \text{cut elimination} & \leftrightarrow & \text{process communication} \end{array}$$

- Type judgments describe the protocols implemented by a process, linear and unrestricted
- Typing rules correspond to the sequent calculus for CLL with mix, plus **new modalities** $\&A$ and $\oplus A$, related by duality
- Key principles:
  - **As before**: Identity as forwarding; cut as parallel + restriction
  - **New**: Monadic approach to non-det via the dual modalities

Types and Duality

Types are assigned to names and describe their session behavior:

$x : A \otimes B$	[Output name of type A, continue as B]
$x : A \parr B$	[Input name of type A, continue as B]
$x : !A$	[Persistent offer of A]
$x : ?A$	[Request of a persistent behavior A]
$x : A \,\&\, B$	[Offer *both* A and B]
$x : A \oplus B$	[Select *either* A or B]
$x : 1$	[Terminated interaction]
$x : \bot$	[Terminated interaction]

Duality of A, written $\overline{A}$, is standard

Types and Duality

Types are assigned to names and describe their session behavior:

$x : A \otimes B$	[Output name of type A, continue as B]
$x : A \parr B$	[Input name of type A, continue as B]
$x : !A$	[Persistent offer of A]
$x : ?A$	[Request of a persistent behavior A]
$x : A \,\&\, B$	[Offer *both* A and B]
$x : A \oplus B$	[Select *either* A or B]
$x : 1$	[Terminated interaction]
$x : \bot$	[Terminated interaction]
$x : \,\&\, A$	**Produce** some non-det behavior of type A on x
$x : \oplus A$	**Consume** any non-det behavior of type A on x

Duality of A, written $\overline{A}$, is standard, with $\overline{\oplus A} = \,\&\,\overline{A}$ and $\overline{\,\&\, A} = \oplus \overline{A}$.

Non Determinism & Failure: Typing Rules

$$\frac{\vdash \Delta, A; \Theta}{\vdash \Delta, \&A; \Theta}\ (\mathsf{T}\&_d^x)$$

$$\frac{}{\vdash \&A; \Theta}\ (\mathsf{T}\&^x)$$

$$\frac{\vdash \&\Delta, A; \Theta}{\vdash \&\Delta, \oplus A; \Theta}\ (\mathsf{T}\oplus_{\vec{w}}^x)$$

$$\frac{\vdash \&\Delta; \Theta \qquad \vdash \&\Delta; \Theta}{\vdash \&\Delta; \Theta}\ (\mathsf{T}\&)$$

Non Determinism & Failure: Typing Rules

$$\frac{P \vdash \Delta, x{:}A; \Theta}{x.\overline{\texttt{some}}; P \vdash \Delta, x{:}\&A; \Theta}\ (\mathsf{T}\&_d^x)$$

$$\frac{}{x.\overline{\texttt{none}} \vdash x{:}\&A; \Theta}\ (\mathsf{T}\&^x)$$

$$\frac{P \vdash \overline{w}{:}\&\Delta, x{:}A; \Theta}{x.\texttt{some}_{\overline{w}}; P \vdash \overline{w}{:}\&\Delta, x{:}\oplus A; \Theta}\ (\mathsf{T}\oplus_{\vec{w}}^x)$$

$$\frac{P \vdash \&\Delta; \Theta \qquad Q \vdash \&\Delta; \Theta}{P \oplus Q \vdash \&\Delta; \Theta}\ (\mathsf{T}\&)$$

Non Determinism & Failure: Semantics (1/3)

[Omitting the exponential context Θ, for simplicity]

$$\dfrac{\dfrac{P \vdash \overline{w}{:}\&\Delta, x{:}A}{x.\mathtt{some}_{\overline{w}}; P \vdash \overline{w}{:}\&\Delta, x{:}\oplus A} \quad \dfrac{Q \vdash \Delta', x{:}\overline{A}}{x.\overline{\mathtt{some}}; Q \vdash \Delta', x{:}\&\overline{A}}}{(\nu x)(x.\mathtt{some}_{\overline{w}}; P \mid x.\overline{\mathtt{some}}; Q) \vdash \overline{w}{:}\&\Delta, \Delta'}$$

Non Determinism & Failure: Semantics (1/3)

[Omitting the exponential context Θ, for simplicity]

$$\dfrac{\dfrac{P \vdash \overline{w}{:}\&\Delta, x{:}A}{x.\mathtt{some}_{\overline{w}}; P \vdash \overline{w}{:}\&\Delta, x{:}\oplus A} \quad \dfrac{Q \vdash \Delta', x{:}\overline{A}}{x.\overline{\mathtt{some}}; Q \vdash \Delta', x{:}\&\overline{A}}}{(\nu x)(x.\mathtt{some}_{\overline{w}}; P \mid x.\overline{\mathtt{some}}; Q) \vdash \overline{w}{:}\&\Delta, \Delta'}$$

$$\longrightarrow$$

$$\dfrac{P \vdash \overline{w}{:}\&\Delta, x{:}A \quad Q \vdash \Delta', x{:}\overline{A}}{(\nu x)(P \mid Q) \vdash \overline{w}{:}\&\Delta, \Delta'}$$

Non Determinism & Failure: Semantics (2/3)

[Omitting the exponential context Θ, for simplicity]

$$\dfrac{\dfrac{P \vdash \overline{w}{:}\&\Delta, x{:}A}{x.\mathtt{some}_{\overline{w}}; P \vdash \overline{w}{:}\&\Delta, x{:}\oplus A} \quad \dfrac{}{x.\overline{\mathtt{none}} \vdash x{:}\&\overline{A}}}{(\nu x)(x.\mathtt{some}_{\overline{w}}; P \mid x.\overline{\mathtt{none}}) \vdash \overline{w}{:}\&\Delta}$$

Non Determinism & Failure: Semantics (2/3)

[Omitting the exponential context Θ, for simplicity]

$$\dfrac{\dfrac{P \vdash \overline{w}{:}\&\Delta, x{:}A}{x.\mathtt{some}_{\overline{w}}; P \vdash \overline{w}{:}\&\Delta, x{:}\oplus A} \quad \dfrac{}{x.\overline{\mathtt{none}} \vdash x{:}\&\overline{A}}}{(\nu x)(x.\mathtt{some}_{\overline{w}}; P \mid x.\overline{\mathtt{none}}) \vdash \overline{w}{:}\&\Delta}$$

$$\longrightarrow$$

$$\dfrac{}{w_1.\overline{\mathtt{none}} \mid \cdots \mid w_n.\overline{\mathtt{none}} \vdash \overline{w}{:}\&\Delta}$$

Non Determinism & Failure: Semantics (3/3)

[Omitting the exponential context Θ, for simplicity]

$$\dfrac{P \vdash \&\Delta, x{:}A \qquad \dfrac{Q \vdash \&\Delta', x{:}\&\overline{A} \qquad R \vdash \&\Delta', x{:}\&\overline{A}}{Q \oplus R \vdash \&\Delta', x{:}\&\overline{A}}}{(\nu x)(P \mid (Q \oplus R)) \vdash \&\Delta, \&\Delta'}$$

Non Determinism & Failure: Semantics (3/3)

[Omitting the exponential context Θ, for simplicity]

$$\dfrac{P \vdash \&\Delta, x{:}A \qquad \dfrac{Q \vdash \&\Delta', x{:}\&\overline{A} \qquad R \vdash \&\Delta', x{:}\&\overline{A}}{Q \oplus R \vdash \&\Delta', x{:}\&\overline{A}}}{(\nu x)(P \mid (Q \oplus R)) \vdash \&\Delta, \&\Delta'}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{P \vdash \&\Delta, x{:}A \qquad Q \vdash \&\Delta', x{:}\&\overline{A}}{(\nu x)(P \mid Q) \vdash \&\Delta, \&\Delta'} \qquad \dfrac{P \vdash \&\Delta, x{:}A \qquad R \vdash \&\Delta', x{:}\&\overline{A}}{(\nu x)(P \mid R) \vdash \&\Delta, \&\Delta'}}{(\nu x)(P \mid Q) \oplus (\nu x)(P \mid R) \vdash \&\Delta, \&\Delta'}$$

- A congruence axiom, rather than a reduction rule
  Non-collapsing: all non-deterministic branches are kept open
- Natural distribution law of parallel over internal non-det choice

Main Results (1/2)

Write $\cong_s$ to denote a process congruence capturing proof conversions (computational, structural, commuting):

Theorem (Cut Elimination)

If $P \vdash \Delta; \Theta$ *then there is a process* Q *such that* $P \cong_s Q$ *and* $Q \vdash \Delta; \Theta$ *is derivable without using rules* ($T\mathsf{cut}$) *and* ($T\mathsf{cut}^?$).

Theorem (Type Preservation)

If $P \vdash \Delta; \Theta$ *and* $P \to Q$ *then* $Q \vdash \Delta; \Theta$.

Write $\mathit{live}(P)$ whenever a prefix (non replicated) is at top-level:

Theorem (Progress)

If $P \vdash ; \Theta$ *and* $\mathit{live}(P)$ *then there is* Q *such that* $P \to Q$.

Also: strong normalization and confluence hold [Pérez et al.'12].

Main Results (2/2)

Write $P \to_c Q$ (and $P \Rightarrow_c Q$) to denote reduction extended with collapsing (non-confluent) rules $P \oplus Q \to P$ and $P \oplus Q \to Q$.

We say P is **prime** if $P \not\equiv (Q \oplus R)$ with non-trivial Q and R.

Theorem (Postponing)

Let $P \vdash \Delta; \Theta$. We have

1. *If $P \Rightarrow P_1 \oplus \ldots \oplus P_n \not\to$ with P_i prime for all i, then $P \Rightarrow_c P_i$ for all $0 < i \leq n$.*
2. *Let $\mathcal{C} = \{\ P_i \mid P \Rightarrow_c P_i \not\to_c$ and P_i prime $\}$. Then $\mathcal{C}$ is finite up to $\equiv$, with $\#\mathcal{C} = n$, and $P \Rightarrow P_1 \oplus \ldots \oplus P_n \to_c P_i$ $(0 < i \leq n)$.*

Outline

Context

Logical Foundations for Concurrency
- Curry-Howard correspondence for Concurrency
- Example: Movie Server

Adding Non Determinism and Failure
- Syntax and Semantics
- Types
- Properties

Examples
- Extended Movie Server
- Encoding λ^{exc}

Concluding Remarks

A Simple Movie Server

We showcase our typed model with a simple movie server process, incrementally featuring:

- Basic session communication and deterministic choices
- Non deterministic choices
- Failure

A Simple Movie Server

$$SBody(s) \triangleq s.\mathtt{case}(s(title).s(card).\overline{s}(movie).s.\overline{\mathtt{close}},\ s(title).\overline{s}(trailer).s.\overline{\mathtt{close}})$$

$$Alice(s) \triangleq s.\mathtt{inr}; \overline{s}(\text{“}Odyssey\text{”}).s(preview).s.\mathtt{close}; 0$$

$$Bob(s) \triangleq s.\mathtt{inl}; \overline{s}(\text{“}Matrix\text{”}).\overline{s}(bobscard).s(mpeg).s.\mathtt{close}; 0$$

$$System_1 \triangleq (\nu s)(SBody(s) \mid Alice(s))$$

Types

Recall that $A \multimap B$ and $\overline{A} \parr B$ are equivalent.

Assuming basic types `Title`, `CCard`, and `Mov` we have:

$$SBT \triangleq (\mathtt{Title} \multimap \mathtt{CCard} \multimap \mathtt{Mov} \otimes 1)\ \&\ (\mathtt{Title} \multimap \mathtt{Mov} \otimes 1)$$

$$SBody(s) \vdash s : SBT\ ; \cdot$$

$$Alice(s) \vdash s : \overline{SBT}\ ; \cdot \qquad Bob(s) \vdash s : \overline{SBT}\ ; \cdot$$

Movie Server, with Infinite Behavior

$$
\begin{aligned}
Movies(srv) &\triangleq !srv(s).SBody(s) \\
Bob(s) &\triangleq s.\mathtt{inl}; \overline{s}(\text{“}Matrix\text{”}).\overline{s}(bobscard).s(mpeg).s.\mathtt{close}; 0 \\
SAlice(srv) &\triangleq \overline{\overline{srv}(s).Alice(s)} \\
SBob(srv) &\triangleq \overline{\overline{srv}(s).Bob(s)} \\
System_2 &\triangleq (\nu srv)(Movies(srv) \mid SAlice(srv) \mid SBob(srv))
\end{aligned}
$$

Types

$$
\begin{aligned}
SAlice(srv) &\vdash \cdot\ ; srv : \overline{\overline{SBT}} \\
SBob(srv) &\vdash \cdot\ ; srv : \overline{\overline{SBT}} \\
SAlice(srv) \mid SBob(srv) &\vdash srv : ?\overline{\overline{SBT}};\ \cdot \\
Movies(srv) &\vdash srv : !SBT\ ;\ \cdot
\end{aligned}
$$

Movie Server, with Non Determinism

A non-deterministic client:

$$Randy(s) \triangleq s.\overline{\texttt{some}}; Alice(s) \oplus s.\overline{\texttt{some}}; Bob(s)$$

$$USystem \triangleq (\nu s)(s.\texttt{some}_{\emptyset}; SBody(s) \mid Randy(s))$$

We derive $USystem \vdash \cdot \,;\, \cdot$

Movie Server, with Non Determinism

A non-deterministic client:

$$\mathit{Randy}(s) \triangleq s.\overline{\texttt{some}}; \mathit{Alice}(s) \oplus s.\overline{\texttt{some}}; \mathit{Bob}(s)$$

$$\mathit{USystem} \triangleq (\nu s)(s.\texttt{some}_{\emptyset}; \mathit{SBody}(s) \mid \mathit{Randy}(s))$$

We derive $\mathit{USystem} \vdash \cdot\ ;\ \cdot$ as follows. First, the non-det client:

$$\dfrac{\dfrac{\mathit{Alice}(s) \vdash s : \overline{SBT}\ ;\ \cdot}{s.\overline{\texttt{some}}; \mathit{Alice}(s) \vdash s : \&\overline{SBT}\ ;\ \cdot} \quad \dfrac{\mathit{Bob}(s) \vdash s : \overline{SBT}\ ;\ \cdot}{s.\overline{\texttt{some}}; \mathit{Bob}(s) \vdash s : \&\overline{SBT}\ ;\ \cdot}}{s.\overline{\texttt{some}}; \mathit{Alice}(s) \oplus s.\overline{\texttt{some}}; \mathit{Bob}(s) \vdash s : \&\overline{SBT}\ ;\ \cdot}$$

Movie Server, with Non Determinism

A non-deterministic client:

$$Randy(s) \triangleq s.\overline{\texttt{some}}; Alice(s) \oplus s.\overline{\texttt{some}}; Bob(s)$$

$$USystem \triangleq (\nu s)(s.\texttt{some}_{\emptyset}; SBody(s) \mid Randy(s))$$

We derive $USystem \vdash \cdot\ ;\ \cdot$ as follows. First, the non-det client:

$$\dfrac{\dfrac{Alice(s) \vdash s : \overline{SBT}\ ;\ \cdot}{s.\overline{\texttt{some}}; Alice(s) \vdash s : \&\overline{SBT}\ ;\ \cdot} \quad \dfrac{Bob(s) \vdash s : \overline{SBT}\ ;\ \cdot}{s.\overline{\texttt{some}}; Bob(s) \vdash s : \&\overline{SBT}\ ;\ \cdot}}{s.\overline{\texttt{some}}; Alice(s) \oplus s.\overline{\texttt{some}}; Bob(s) \vdash s : \&\overline{SBT}\ ;\ \cdot}$$

Then we compose the non-det client with the (adapted) server:

$$\dfrac{\dfrac{SBody(s) \vdash s : SBT\ ;\cdot}{s.\texttt{some}_{\emptyset}; SBody(s) \vdash s : \oplus SBT\ ;\ \cdot} \quad \dfrac{}{Randy(s) \vdash s : \&\overline{SBT}\ ;\ \cdot}}{USystem \vdash \cdot\ ;\ \cdot}$$

Movie Server, with NonDet and a Boolean log

A server with a log, with type $\mathrm{B} = 1 \oplus 1$:

$$\begin{aligned} \mathit{SBodyL}(s) \triangleq\ & s.\mathtt{case}(s(\mathit{title}).\overline{s}(\mathit{trailer}).s.\overline{\mathtt{close}} \mid l.\mathtt{inl}; l.\overline{\mathtt{close}}, \\ & s(\mathit{tit}).s(\mathit{card}).\overline{s}(\mathit{mov}).s.\overline{\mathtt{close}} \mid l.\mathtt{inr}; l.\overline{\mathtt{close}}\,) \end{aligned}$$

We have $\mathit{SBodyL}(s) \vdash s{:}\mathit{SBT}, l{:}\mathrm{B}\ ;\ \cdot$.

But this can't be composed with client $\mathit{Randy}(s) \vdash s : \&\overline{\mathit{SBT}}\ ;\ \cdot$.

Movie Server, with NonDet and a Boolean log

A server with a log, with type $\mathrm{B} = 1 \oplus 1$:

$$SBodyL(s) \triangleq s.\mathtt{case}(s(title).\overline{s}(trailer).s.\overline{\mathtt{close}} \mid l.\mathtt{inl}; l.\overline{\mathtt{close}},$$
$$s(tit).s(card).\overline{s}(mov).s.\overline{\mathtt{close}} \mid l.\mathtt{inr}; l.\overline{\mathtt{close}}\,)$$

We have $SBodyL(s) \vdash s{:}SBT, l{:}\mathrm{B}\ ;\cdot$.

But this can't be composed with client $Randy(s) \vdash s : \&\overline{SBT}\ ;\ \cdot$.

Rather, we could have:

$$s.\mathtt{some}_l; l.\overline{\mathtt{some}}; SBodyL(s) \vdash s{:}\oplus SBT, l{:}\&\mathrm{B}\ ;\cdot$$

and derive:

$$ULSystem \triangleq (\nu s)(s.\mathtt{some}; l.\overline{\mathtt{some}}; SBodyL(s) \mid Randy(s))$$
$$ULSystem \vdash l{:}\&\mathrm{B}\ ;\cdot$$

Movie Server, with Failure

A faulty client:

$$Buzz(s) \triangleq s.\overline{\texttt{some}}; Bob(s) \oplus s.\overline{\texttt{none}}$$

Movie Server, with Failure

A faulty client:

$$Buzz(s) \triangleq s.\overline{\texttt{some}}; Bob(s) \oplus s.\overline{\texttt{none}}$$

It has the same type as the non-det client $Randy(s)$:

$$Buzz(s) \vdash s : \&\overline{SBT} \; ; \cdot$$

and so we can compose it with the server with log:

$$(\nu s)(SBodyL(s) \mid Buzz(s)) \vdash l{:}\&\mathrm{B} \; ; \cdot$$

Movie Server, with Failure

A faulty client:

$$Buzz(s) \triangleq s.\overline{\texttt{some}}; Bob(s) \oplus s.\overline{\texttt{none}}$$

It has the same type as the non-det client $Randy(s)$:

$$Buzz(s) \vdash s : \&\overline{SBT} \; ; \cdot$$

and so we can compose it with the server with log:

$$(\nu s)(SBodyL(s) \mid Buzz(s)) \vdash l{:}\&\mathrm{B} \; ; \cdot$$

Failure of sub-computations propagates inside the monad $\&-$.
We have the reduction sequence:

$$(\nu s)(SBodyL(s) \mid Buzz(s)) \Rightarrow (l.\overline{\texttt{none}} \oplus l.\texttt{inl}; l.\overline{\texttt{close}})$$

The system thus either aborts or chooses $l.\texttt{inl}$ on the log.

λ^{exc}: Functions, Sessions, Threads & Non-det

An extended case study for the expressiveness of our type system.

Expressions

$$
\begin{array}{rcl}
v & ::= & x \mid * \mid \lambda z.e \mid \langle\!\langle v \rangle\!\rangle \\
e & ::= & v \mid (f\ x) \mid \texttt{LET}\ a = e_1\ \texttt{IN}\ e_2 \\
& \mid & \texttt{TRY}\ e_1\ \texttt{CATCH}\ z.\ e_2 \mid \texttt{THROW}\ z \\
& \mid & \texttt{LIFT}\ e \mid \texttt{SOME}!\, z; e \mid \texttt{SOME}?\, z; e \mid \texttt{NONE}!\, z; e \mid e_1 \oplus e_2 \\
& \mid & \texttt{FORK}\ c.e \mid \texttt{SEND}(c, e_1); e_2 \mid \texttt{RECV}(c, z); e \\
& \mid & \texttt{CLOSE}!\, c\,; e \mid \texttt{CLOSE}?\, c
\end{array}
$$

Types

Expressions are **effectful** (if they can raise an exception) or **pure**

$$
\begin{array}{rcl}
T & ::= & \mathbf{unit} \mid A \xrightarrow{T} B \mid A \xrightarrow{0} B \\
& \mid & !T.T' \mid ?T.T' \mid \mathsf{end}_! \mid \mathsf{end}_? \mid \oplus T \mid \& T
\end{array}
$$

$A \xrightarrow{T} B$ types functions that may raise an exception of type T.
$A \xrightarrow{0} B$ is the type of pure functions.

Types for λ^{exc}

- A type-and-effect system: effects represent the type of the exception that can be raised
- Judgments are then the form

$$\mathsf{D} \vdash^{U} e : T$$

Under environment D, expression e has return type T, while the effect type U is 0 (e is pure) or T' (the type of exceptions)

Sample Typing Rules for λ^{exc}

$$\text{LET1}\quad \frac{\mathsf{D}_1 \vdash^T e_1 : A \qquad \oplus \mathsf{D}_2, a{:}A \vdash^T e_2 : B}{\mathsf{D}_1, \oplus \mathsf{D}_2 \vdash^T \mathtt{LET}\, a = e_1\, \mathtt{IN}\, e_2 : B}$$

$$\text{LET2}\quad \frac{\mathsf{D}_1 \vdash^0 e_1 : A \qquad \mathsf{D}_2, a{:}A \vdash^0 e_2 : B}{\mathsf{D}_1, \mathsf{D}_2 \vdash^0 \mathtt{LET}\, a = e_1\, \mathtt{IN}\, e_2 : B}$$

$$\text{LET3}\quad \frac{\mathsf{D}_1 \vdash^0 e_1 : A \qquad \mathsf{D}_2, a{:}A \vdash^T e_2{:}B}{\mathsf{D}_1, \mathsf{D}_2 \vdash^T \mathtt{LET}\, a = e_1\, \mathtt{IN}\, e_2 : B}$$

$$\text{TRY}\quad \frac{\mathsf{D}_1 \vdash^U e_1 : T \qquad x : U \vdash^0 e_2 : T}{\mathsf{D}_1 \vdash^0 \mathtt{TRY}\, e_1\, \mathtt{CATCH}\, x.e_2 : T} \qquad \text{THROW}\quad \frac{\mathsf{D} \vdash^0 z : T}{\mathsf{D} \vdash^T \mathtt{THROW}\, z : U}$$

λ^{exc} into Typed Processes: Examples (1/2)

In $[\![e]\!]_{y,x}$, name y is a **non-deterministic continuation** where values may be produced; name x denotes the **enclosing handler**.

- Variables

$$[\![z]\!]_y = y\langle z\rangle.y.\overline{\texttt{close}}$$
$$[\![z]\!]_{y,x} = y.\overline{\texttt{some}}; y\langle z\rangle.y\langle x\rangle.y.\overline{\texttt{close}}$$

λ^{exc} into Typed Processes: Examples (1/2)

In $[\![e]\!]_{y,x}$, name y is a **non-deterministic continuation** where values may be produced; name x denotes the **enclosing handler**.

- Variables

$$[\![z]\!]_y = y\langle z\rangle.y.\overline{\texttt{close}}$$
$$[\![z]\!]_{y,x} = y.\overline{\texttt{some}}; y\langle z\rangle.y\langle x\rangle.y.\overline{\texttt{close}}$$

- Let expressions

$$[\![\texttt{LET}\ a = e_1\ \texttt{IN}\ e_2]\!]_y = (\nu q)([\![e_1]\!]_q \mid q(a).q.\texttt{close}; [\![e_2]\!]_y)$$
$$[\![\texttt{LET}\ a = e_1\ \texttt{IN}\ e_2]\!]_{y,x} = (\nu q)([\![e_1]\!]_{q,x} \mid q.\texttt{some}_D; q(a).q(s).q.\texttt{close}; [\![e_2]\!]_{y,s})$$

λ^{exc} into Typed Processes: Examples (2/2)

- Try-catch

$$[\![\texttt{TRY}\ e_1\ \texttt{CATCH}\ z.\ e_2]\!]_y =$$
$$(\nu j)\Big((\nu k)\big([\![e_1]\!]_{k,j} \mid \underbrace{k.\texttt{some}_\emptyset; k(u).k(z).k.\texttt{close}; z.\texttt{inl}; z\langle u\rangle.z.\overline{\texttt{close}}}_{\text{(I)}}\big) \mid \underbrace{j.\texttt{case}(j(u).j.\texttt{close}; y\langle u\rangle.y.\overline{\texttt{close}}\ ,\ j(z).j.\texttt{close}; [\![e_2]\!]_y)}_{\text{(II)}}\Big)$$

where (I) deals with normal behavior,
while (II) handles both normal and exceptional behaviors.

- Throw:

$$[\![\texttt{THROW}\ z]\!]_{y,x} = y.\overline{\texttt{none}} \mid x.\texttt{inr}; x\langle z\rangle.x.\overline{0}$$

Additional Results

(Joint work with Joseph Paulus and Daniele Nantes)

- $\lambda^{\lightning}_{\oplus}$: a resource λ-calculus with non-determinism, bags, and an explicit term denoting failures.
- The syntax and semantics of $\lambda^{\lightning}_{\oplus}$ are based on existing resource λ-calculi
  - Boudol and Laneve (2000)
  - Pagani and Ronchi della Rocca (2010)

  while adopting non-collapsing determinism.
- An **intersection type system** to "count" resources in bags.
- We have encoded $\lambda^{\lightning}_{\oplus}$ into the session π-calculus just presented.
  - The encoding considers **well formed** expressions in $\lambda^{\lightning}_{\oplus}$: well-typed expressions *plus* expressions that lead to failure.
  - The encoding preserves typing and satisfies operational correspondences.

Outline

Context

Logical Foundations for Concurrency
- Curry-Howard correspondence for Concurrency
- Example: Movie Server

Adding Non Determinism and Failure
- Syntax and Semantics
- Types
- Properties

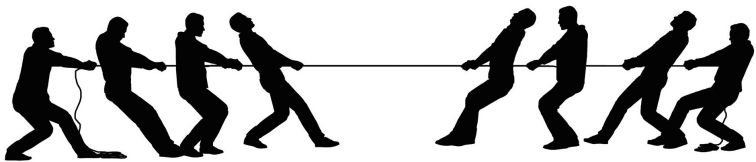
Examples
- Extended Movie Server
- Encoding λ^{exc}

Concluding Remarks

Concluding Remarks

- A Curry-Howard correspondence for concurrency, exploiting linear logic and name-passing processes under session types
- Strong correctness properties on protocol conformance follow from logical principles (e.g., cut elimination)
- Excellent basis for extensions, via developments in logic

Concluding Remarks



- Tackling stateful concurrent programs with linearity, based on a Curry-Howard interpretation of extended CLL as session types
- Our type system supports non-determinism and failure within concurrent programs by exploiting new modalities $\&A$ and $\oplus A$
- Strong correctness properties (e.g. global progress); non-determinism compatible with process calculi approaches

Logical Foundations for Concurrency: Non-Determinism and Failures

Jorge A. Pérez
University of Groningen, The Netherlands
j.a.perez [[at]] rug.nl

ECI 2026
(Version of 31st July 2026)