# LINEAR LOGIC AND BEHAVIORAL TYPES

Luís Caires
Universidade Nova de Lisboa
(based on joint work with Pfenning, Toninho, and Perez)

NOVA Laboratory for
Computer Science and Informatics

EU TYPES 2016 Meeting (NOVA Lisbon)

## type systems for programming

- Types at the heart of "real" PLs (OCaml, Java, C#, Scala)
- Highly modular, based on a "lego" of canonical constructions
- Deep foundations in logic
  - a type system is (should be) a specialised logic!
- Essential in mainstream technology
  - mostly checked statically
  - "standard" types must be easy to use by any programmer
  - types are everywhere, and "practical"

## type systems for programming

- Huge impact on software quality:
  - “Well-typed programs do not go wrong”
- Huge impact on programming (as a human activity):
  - types “tame programmers” to write reasonable code
- But what about types (specifically) for concurrency ?
  - “Adopted” type systems are purely structural, state oblivious, unable to tackle the challenges of state dynamics, concurrency, aliasing, etc (but recently, see e.g,, Rust).
  - We foresee behavioural types leading to a new generation of type systems for future programming languages

## simply typed λ-calculus [Church30]

$\Gamma \vdash M : A$

$$\Gamma, x{:}A \vdash x{:}A$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$\texttt{Tapp}(\texttt{Tlam}([x]d_1), d_2) \to d_1\{d_2/x\}$

## simply typed λ-calculus

$$\Gamma \vdash * : 1$$

$$\frac{\Gamma \vdash M : \mathrm{A} \quad \Gamma \vdash N : \mathrm{B}}{\Gamma \vdash \langle M, N \rangle : \mathrm{A} \wedge \mathrm{B}}$$

$$\frac{\Gamma \vdash M : \mathrm{A} \wedge \mathrm{B}}{\Gamma \vdash \mathsf{fst}(M) : \mathrm{A}} \qquad \mathtt{Tfst}(\mathtt{Tpair}(d_1, d_2)) \to d_1$$

$$\frac{\Gamma \vdash M : \mathrm{A} \wedge \mathrm{B}}{\Gamma \vdash \mathsf{snd}(M) : \mathrm{B}} \qquad \mathtt{Tsnd}(\mathtt{Tpair}(d_1, d_2)) \to d_2$$

## simply typed λ-calculus

$$\frac{\Gamma \vdash M : \mathrm{A}}{\Gamma \vdash \mathtt{inl}(M) : \mathrm{A} \vee \mathrm{B}}$$

$\mathtt{Tcase}(\mathtt{Tinl}(d), [x]c_1, [x]c_2) \rightarrow c_1\{d/x\}$

$$\frac{\Gamma \vdash M : \mathrm{B}}{\Gamma \vdash \mathtt{inr}(M) : \mathrm{A} \vee \mathrm{B}}$$

$\mathtt{Tcase}(\mathtt{Tinr}(d), [x]c_1, [x]c_2) \rightarrow c_2\{d/x\}$

$$\frac{\Gamma \vdash N : \mathrm{A} \vee \mathrm{B} \quad \Gamma, x{:}\mathrm{A} \vdash M : \mathrm{C} \quad \Gamma, x{:}\mathrm{B} \vdash N : \mathrm{C}}{\Gamma \vdash \mathtt{case}\ N\ (\mathtt{inl}(x) \Rightarrow M \mid \mathtt{inr}(x) \Rightarrow N) : \mathrm{C}}$$

## induction

$$\Gamma \vdash \mathsf{nil} : \mathsf{List}[A]$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : \mathsf{List}[A]}{\Gamma \vdash M{::}N : \mathsf{List}[A]}$$

$$\frac{\Gamma \vdash N : C \qquad \Gamma, x{:}A, t{:}\ \mathsf{List}[A], z : C \vdash M : C}{\Gamma \vdash \mathsf{rec}\ (0 \Rightarrow N, (x,t,z)M) : C}$$

## Basic Properties of Typing

- type preservation under evaluation / reduction
  - think about rewriting the complete typing trees
- progress (stuck freedom)
  - together with preservation this means “type safety”
- termination (sometimes)
- confluence (sometimes)

## Typeful Programming [Cardelli85]

- Typeful programming ~ Special case of program specification
- Types ~ Specifications
- Type-Checking ~ Verification
- Useful to enforce correctness at "compilation" time
- View nicely fits with the Curry-Howard paradigm of propositions as types, and proofs as programs

## Propositions as Types

# Intuitionistic Logic = Typed λ-calculus

$$\Gamma, x{:}A \vdash x{:}A \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\Gamma \vdash {*}{:}1 \qquad \frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash \mathtt{snd}(M) : B} \qquad \frac{\Gamma \vdash M : A \wedge B}{\Gamma \vdash \mathtt{fst}(M) : A} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M,N \rangle : A \wedge B}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathtt{inl}(M) : A \vee B} \qquad \frac{\Gamma \vdash N : A \vee B \quad \Gamma, x{:}A \vdash M : C \; \Gamma, x{:}B \vdash M : C}{\Gamma \vdash \mathtt{case}\; N\, (\mathtt{inl}(x) \Rightarrow M \mid \mathtt{inr}(x) \Rightarrow M) : C}$$

$$\frac{\Gamma \vdash M : B}{\Gamma \vdash \mathtt{inr}(M) : A \vee B}$$

# Curry-Howard Correpondence

- Proofs = Programs and Types = Propositions
- Curry-Howard, Girard, Wadler
- A proof denotes a "computational object": program, process
- Program execution = Proof reduction (cut-elimination)
- Program equivalence = Proof conversion
- Proof reduction preserves proof equivalence
- Termination + Confluence = Consistency
- Compositional Semantics via Logical Relations

## Curry-Howard Design Space

- Different logics yield different typed languages
  - Sequent calculus $\leadsto$ explicit substitutions
  - Higher order logic $\leadsto$ polymorphism
  - Classical logic $\leadsto$ continuations, exceptions
  - Modal logic $\leadsto$ monads, security
  - Linear Logic $\leadsto$ resource control, behavioural types
- “***Powerful insights arise from linking two fields of study previously thought separate [ … ] as offered by the principle of Propositions as Types, which links logic to computation. At first sight it appears to be a simple coincidence— almost a pun—but it turns out to be remarkably robust, inspiring the design of automated proof assistants and programming languages***” [Wadler16]

## Curry-Howard Design Space

- Different logics yield different typed languages
  - Sequent calculus $\rightsquigarrow$ explicit substitutions
  - Higher order logic $\rightsquigarrow$ polymorphism
  - Classical logic $\rightsquigarrow$ continuations, exceptions
  - Modal logic $\rightsquigarrow$ monads, security
  - Linear Logic $\rightsquigarrow$ resource control, behavioural types
- ***"One can also extrapolate this correspondence and turn it into a predictive tool: if a concept is present in type theory but absent in programming, or vice versa, it can be very fruitful to both areas to investigate and see what the corresponding concept might be in the other context."*** [Cardelli89]

## Curry Howard for Process Types?

## Types for Processes

# the $\pi$-calculus [Milner92]

| | | | | |
|---|---|---|---|---|
| $P$ ::= | $\mathbf{0}$ | (inaction) | | |
| \| | $P \mid Q$ | (composition) | | |
| \| | $(\mathsf{new}\ x)P$ | (restriction) | | |
| \| | $x(y).P$ | (input) | | |
| \| | $x[y].P$ | (free output) | | |
| \| | $!x(y).P$ | (replicated input) | | |
| \| | $\bar{x}(y).P$ | $:= (\mathsf{new}\ y)x[y].P$ | | (fresh output) |

## Semantics:

structural congruence ($P \equiv Q$) [ static identity ]
reduction ($P \to Q$) [ dynamics ]

## the $\pi$-calculus [Milner92]

structural congruence (≡)

$P \mid 0 \equiv P$
$P \mid Q \equiv Q \mid P$
$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$

$(\mathsf{new}\ x)0 \equiv 0$
$(\mathsf{new}\ x)(P \mid Q) \equiv P \mid (\mathsf{new}\ x)Q\ [\ x \notin fn(P)\ ]$

## the $\pi$-calculus [Milner92]

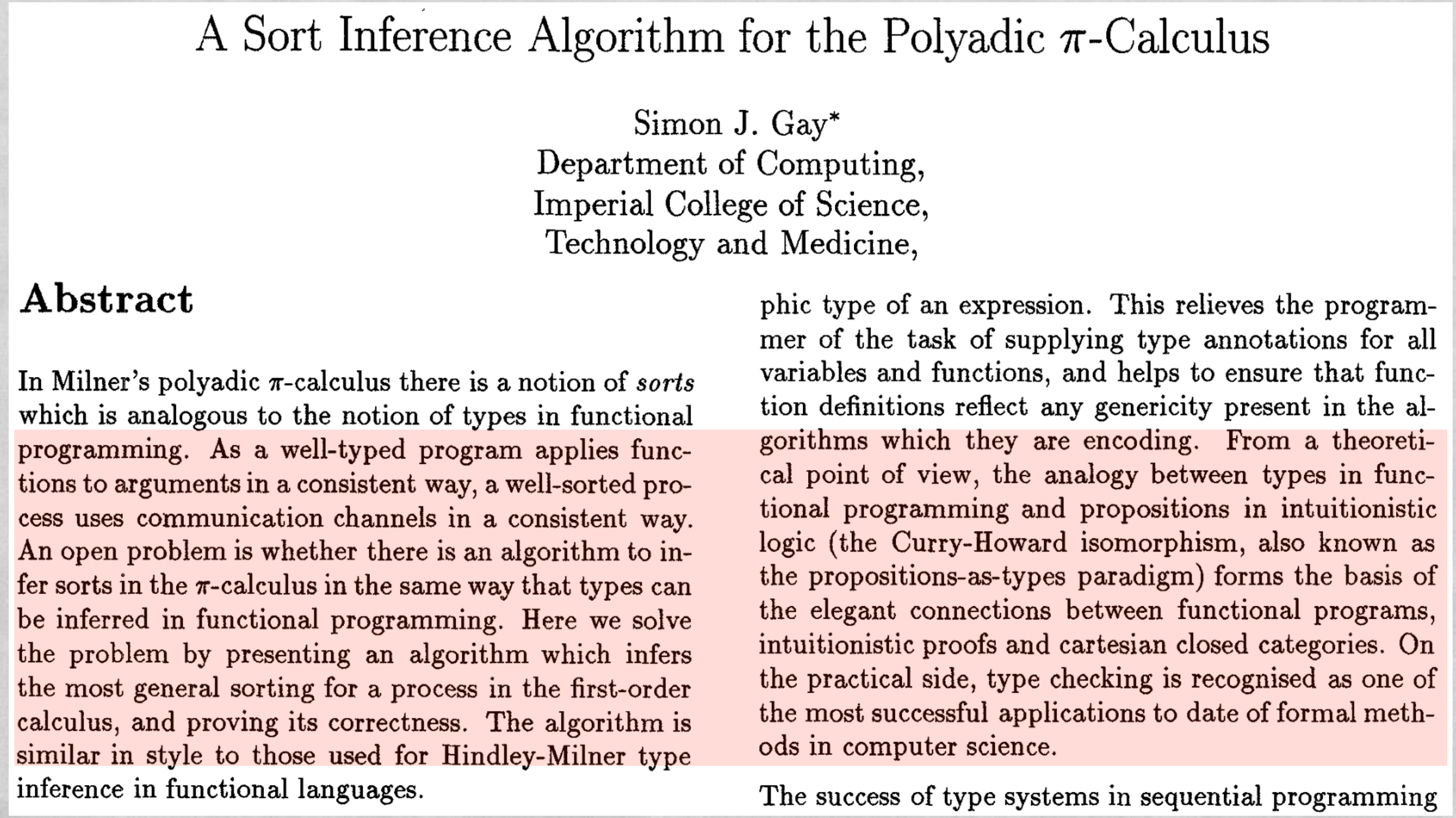
reduction ($\rightarrow$)

$$x(y).P \mid x[z].Q \quad \rightarrow \quad P\{z/y\} \mid Q$$
$$!x(y).P \mid x[z].Q \quad \rightarrow \quad !x(y).P \mid P\{z/y\} \mid Q$$

$$P \rightarrow Q \quad \textit{implies} \quad P \mid R \rightarrow Q \mid R$$
$$P \rightarrow Q \quad \textit{implies} \quad (\text{new } x)P \rightarrow (\text{new } x)Q$$
$$(P \equiv P' \textit{ and } P' \rightarrow Q' \textit{ and } Q' \equiv Q) \textit{ implies } P \rightarrow Q$$

## Types for Processes

# simple types [Milner92,Gay93]



A Sort Inference Algorithm for the Polyadic $\pi$-Calculus

Simon J. Gay*
Department of Computing,
Imperial College of Science,
Technology and Medicine,

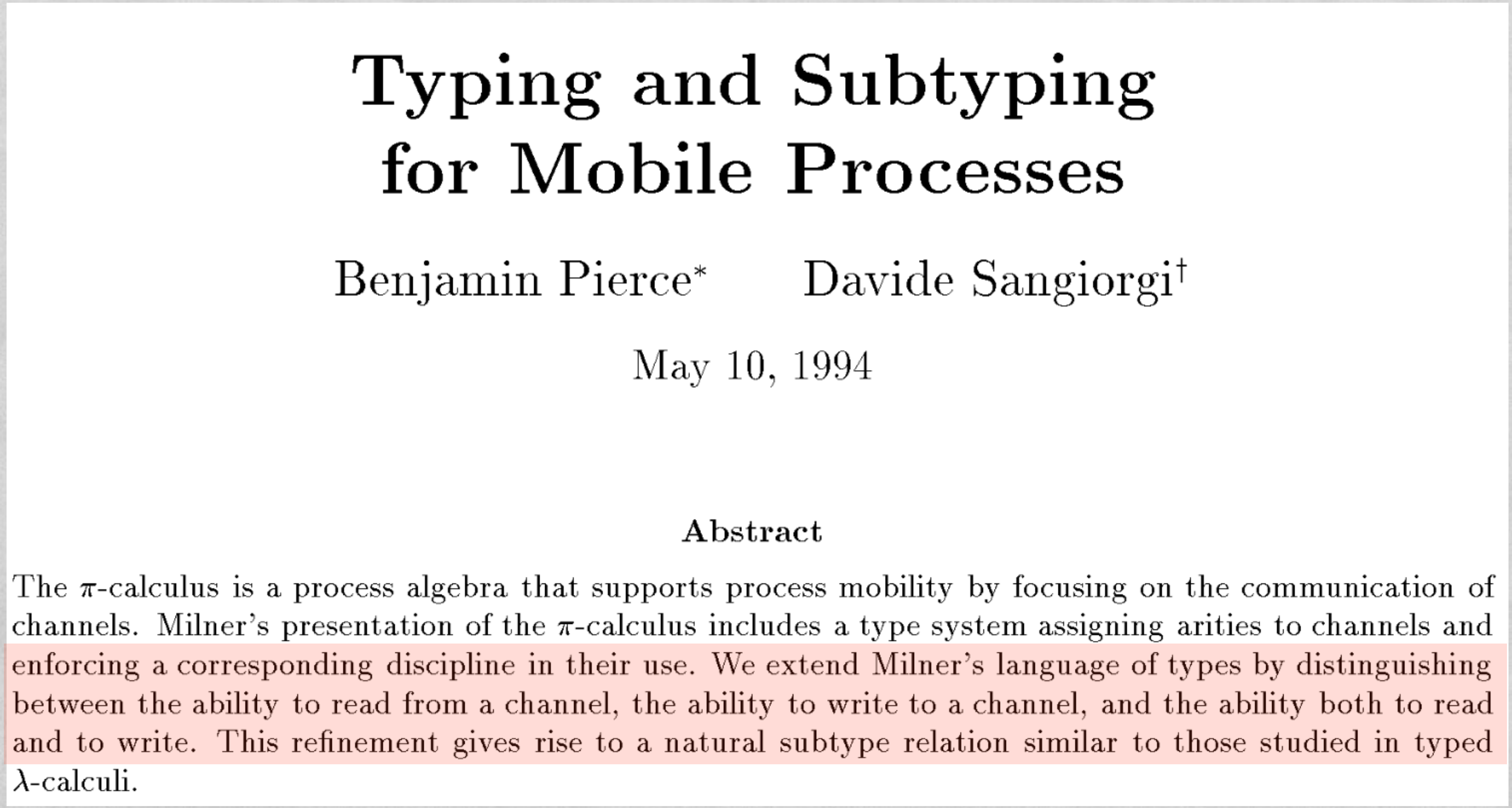
**Abstract**

In Milner's polyadic $\pi$-calculus there is a notion of *sorts* which is analogous to the notion of types in functional programming. As a well-typed program applies functions to arguments in a consistent way, a well-sorted process uses communication channels in a consistent way. An open problem is whether there is an algorithm to infer sorts in the $\pi$-calculus in the same way that types can be inferred in functional programming. Here we solve the problem by presenting an algorithm which infers the most general sorting for a process in the first-order calculus, and proving its correctness. The algorithm is similar in style to those used for Hindley-Milner type inference in functional languages.

phic type of an expression. This relieves the programmer of the task of supplying type annotations for all variables and functions, and helps to ensure that function definitions reflect any genericity present in the algorithms which they are encoding. From a theoretical point of view, the analogy between types in functional programming and propositions in intuitionistic logic (the Curry-Howard isomorphism, also known as the propositions-as-types paradigm) forms the basis of the elegant connections between functional programs, intuitionistic proofs and cartesian closed categories. On the practical side, type checking is recognised as one of the most successful applications to date of formal methods in computer science.

The success of type systems in sequential programming

# IO-types [PierceSangiorgi93]



## Typing and Subtyping for Mobile Processes

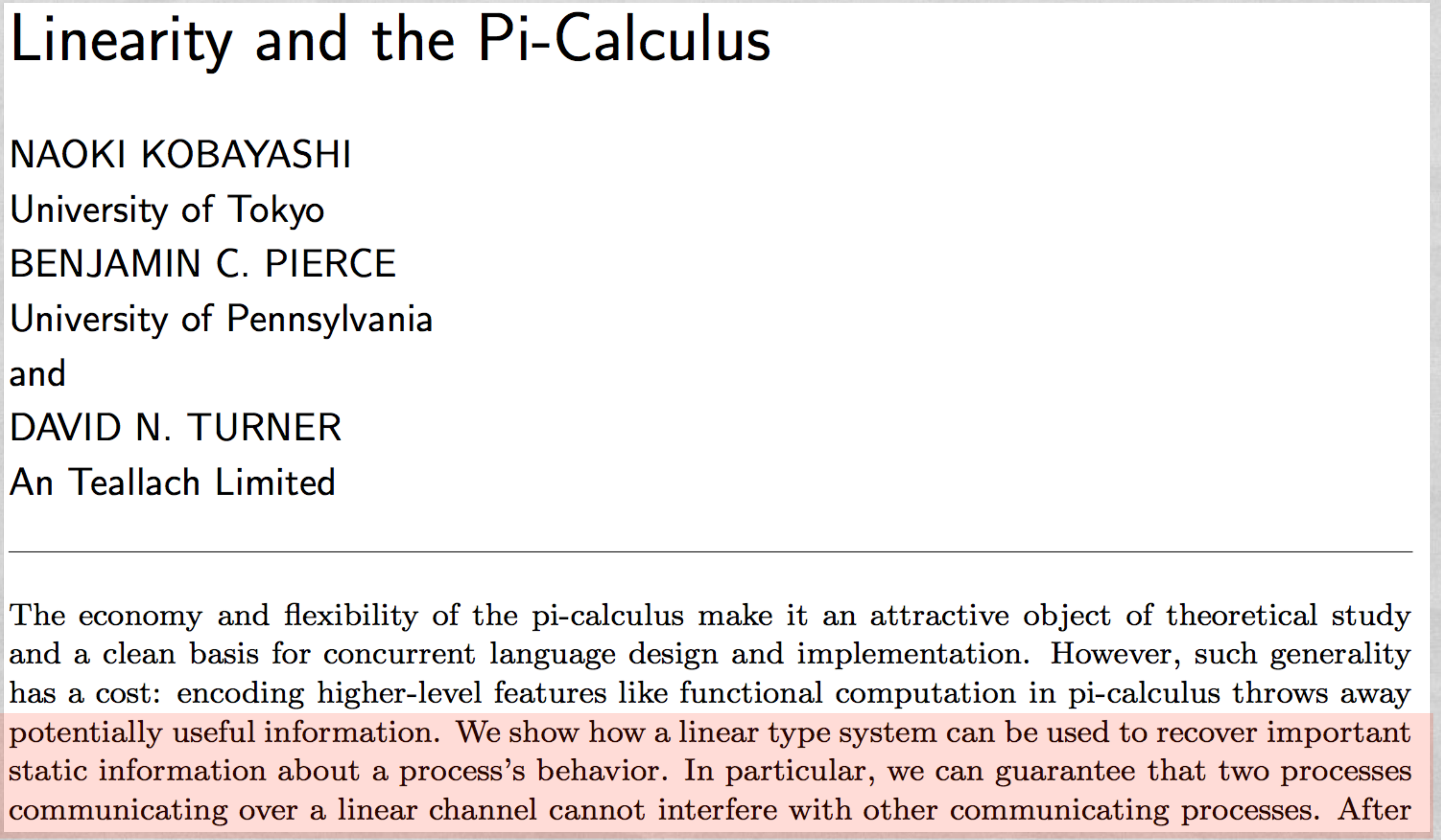
Benjamin Pierce[*] Davide Sangiorgi[†]

May 10, 1994

**Abstract**

The $\pi$-calculus is a process algebra that supports process mobility by focusing on the communication of channels. Milner's presentation of the $\pi$-calculus includes a type system assigning arities to channels and enforcing a corresponding discipline in their use. We extend Milner's language of types by distinguishing between the ability to read from a channel, the ability to write to a channel, and the ability both to read and to write. This refinement gives rise to a natural subtype relation similar to those studied in typed $\lambda$-calculi.

# Linear Types [KobayashiPierceTurner96]

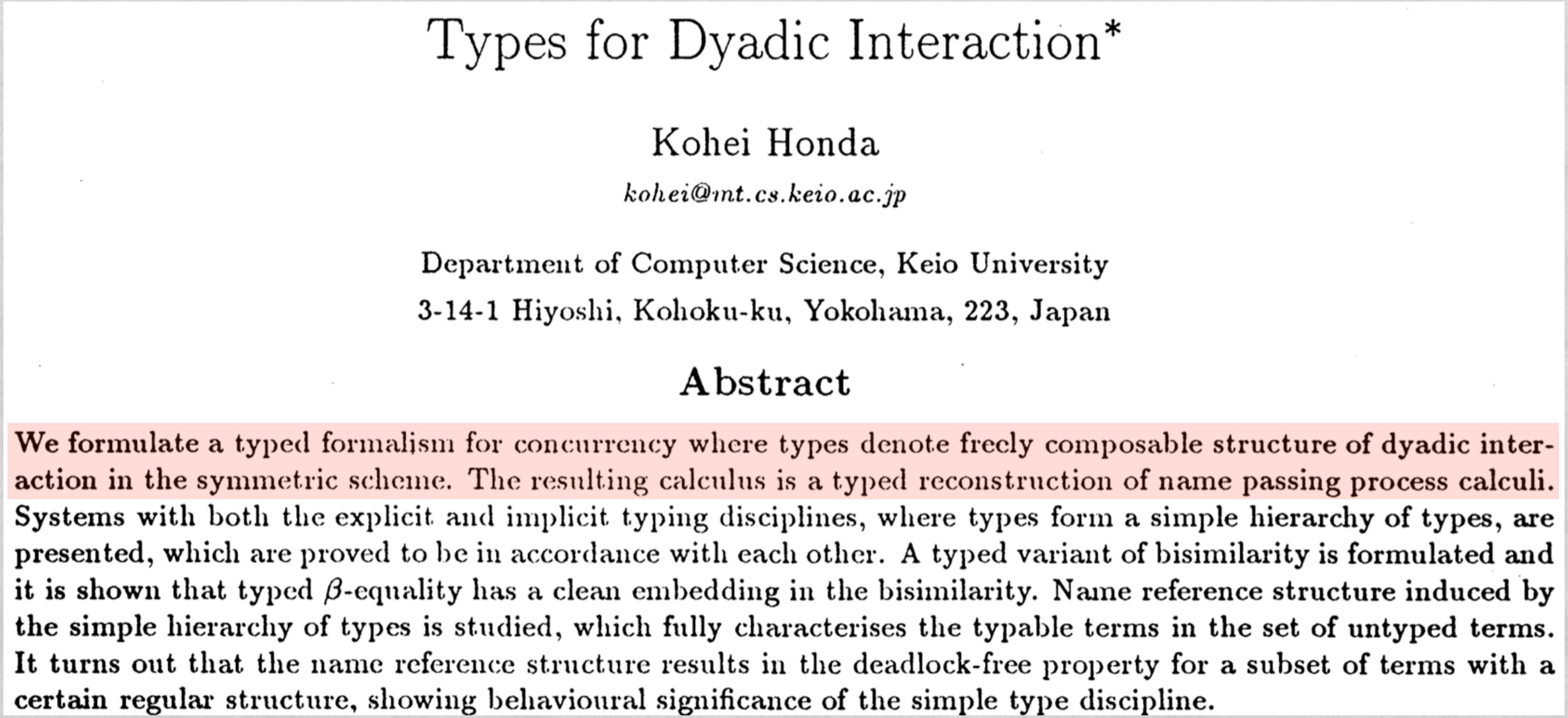


Linearity and the Pi-Calculus

NAOKI KOBAYASHI
University of Tokyo
BENJAMIN C. PIERCE
University of Pennsylvania
and
DAVID N. TURNER
An Teallach Limited

The economy and flexibility of the pi-calculus make it an attractive object of theoretical study and a clean basis for concurrent language design and implementation. However, such generality has a cost: encoding higher-level features like functional computation in pi-calculus throws away potentially useful information. We show how a linear type system can be used to recover important static information about a process's behavior. In particular, we can guarantee that two processes communicating over a linear channel cannot interfere with other communicating processes. After

# Session Types [Honda93,HKV98,GH05]



Types for Dyadic Interaction*

Kohei Honda

*kohei@mt.cs.keio.ac.jp*

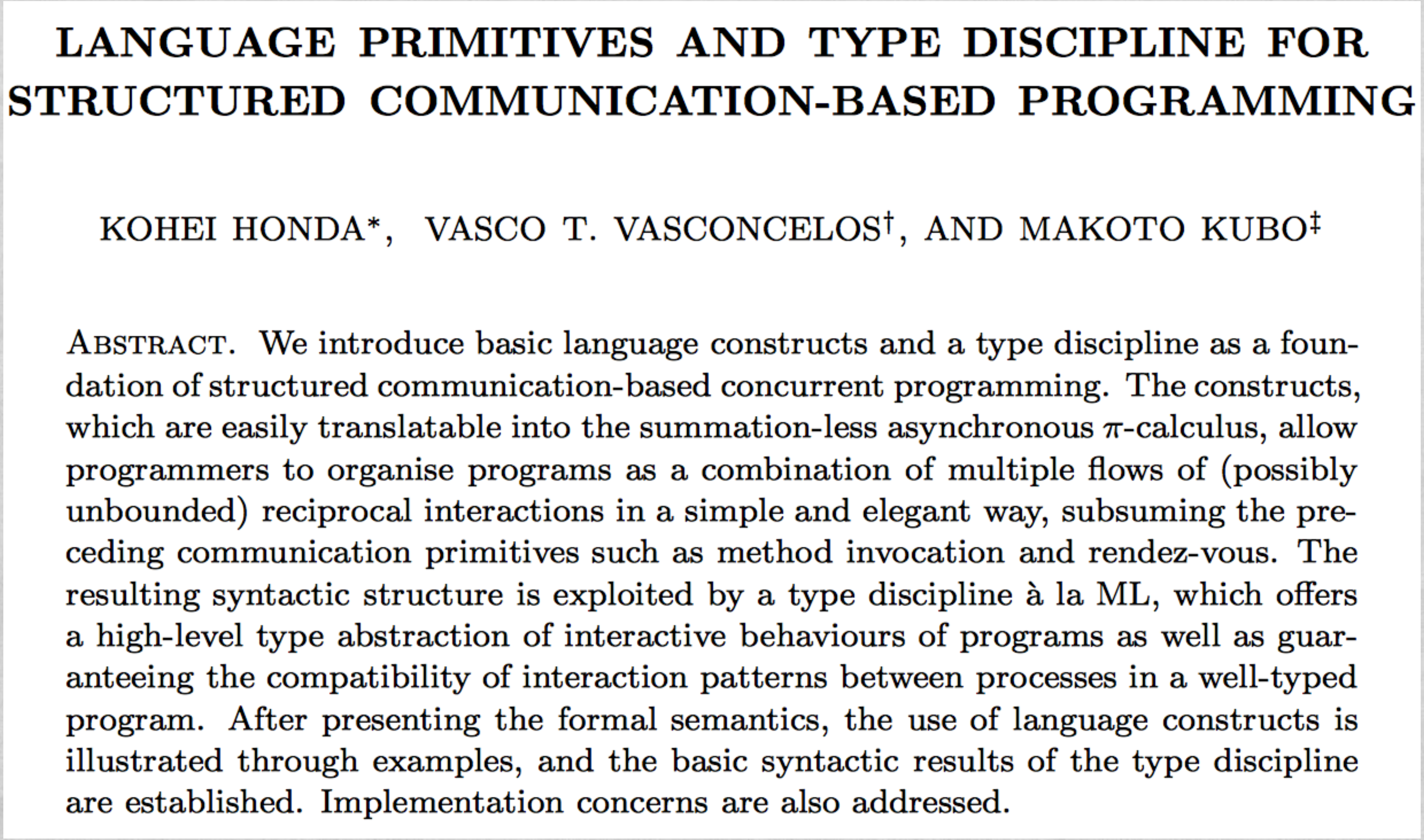
Department of Computer Science, Keio University

3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223, Japan

**Abstract**

We formulate a typed formalism for concurrency where types denote freely composable structure of dyadic interaction in the symmetric scheme. The resulting calculus is a typed reconstruction of name passing process calculi. Systems with both the explicit and implicit typing disciplines, where types form a simple hierarchy of types, are presented, which are proved to be in accordance with each other. A typed variant of bisimilarity is formulated and it is shown that typed $\beta$-equality has a clean embedding in the bisimilarity. Name reference structure induced by the simple hierarchy of types is studied, which fully characterises the typable terms in the set of untyped terms. It turns out that the name reference structure results in the deadlock-free property for a subset of terms with a certain regular structure, showing behavioural significance of the simple type discipline.

# Session Types [Honda93,HKV98,GH05]

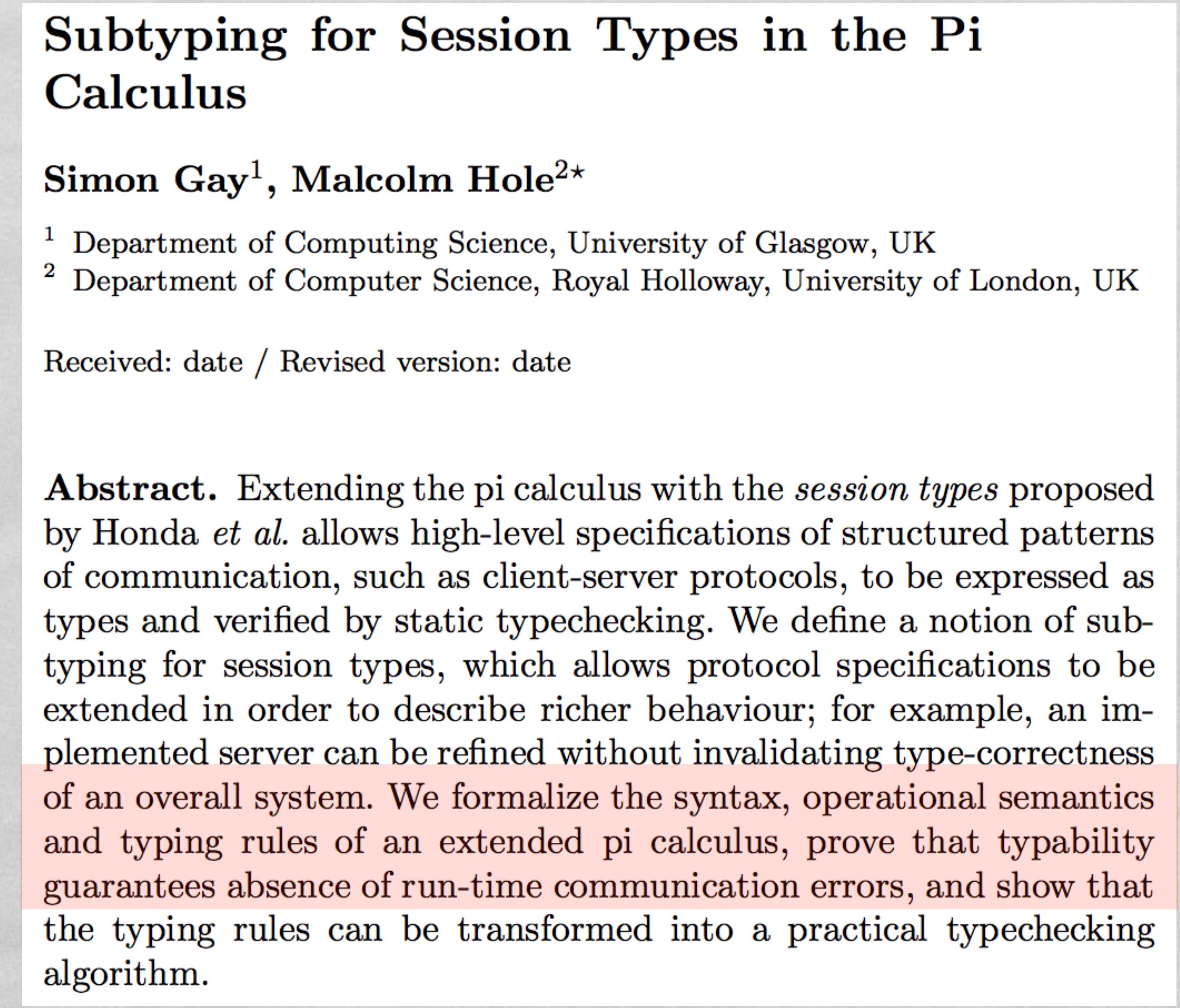


LANGUAGE PRIMITIVES AND TYPE DISCIPLINE FOR STRUCTURED COMMUNICATION-BASED PROGRAMMING

KOHEI HONDA*, VASCO T. VASCONCELOS†, AND MAKOTO KUBO‡

ABSTRACT. We introduce basic language constructs and a type discipline as a foundation of structured communication-based concurrent programming. The constructs, which are easily translatable into the summation-less asynchronous $\pi$-calculus, allow programmers to organise programs as a combination of multiple flows of (possibly unbounded) reciprocal interactions in a simple and elegant way, subsuming the preceding communication primitives such as method invocation and rendez-vous. The resulting syntactic structure is exploited by a type discipline à la ML, which offers a high-level type abstraction of interactive behaviours of programs as well as guaranteeing the compatibility of interaction patterns between processes in a well-typed program. After presenting the formal semantics, the use of language constructs is illustrated through examples, and the basic syntactic results of the type discipline are established. Implementation concerns are also addressed.

## Session Types [Honda93,HKV98,GH05]



### Subtyping for Session Types in the Pi Calculus

**Simon Gay[1], Malcolm Hole[2]***

[1] Department of Computing Science, University of Glasgow, UK
[2] Department of Computer Science, Royal Holloway, University of London, UK

Received: date / Revised version: date

**Abstract.** Extending the pi calculus with the *session types* proposed by Honda *et al.* allows high-level specifications of structured patterns of communication, such as client-server protocols, to be expressed as types and verified by static typechecking. We define a notion of subtyping for session types, which allows protocol specifications to be extended in order to describe richer behaviour; for example, an implemented server can be refined without invalidating type-correctness of an overall system. We formalize the syntax, operational semantics and typing rules of an extended pi calculus, prove that typability guarantees absence of run-time communication errors, and show that the typing rules can be transformed into a practical typechecking algorithm.

# Session Types [GH05]

| | | |
|---|---|---|
| T ::= | *T | (shared channel) |
| \| | S | (session type) |

| | | |
|---|---|---|
| S ::= | end | (base type) |
| \| | !T.S | (output) |
| \| | ?T.S | (input) |

$$\frac{\Gamma; \Delta_1 \vdash P \quad \Gamma;\Delta_2 \vdash Q}{\Gamma; \Delta_1, \Delta_2 \vdash P \mid Q} \qquad \frac{\Gamma; \Delta, x^{+}{:}S, x^{-}{:}\overline{S} \vdash P}{\Gamma; \Delta \vdash (\mathsf{new}\ x)P} \qquad \frac{\Gamma; \cdot \vdash P}{\Gamma; \cdot \vdash\ !P}$$

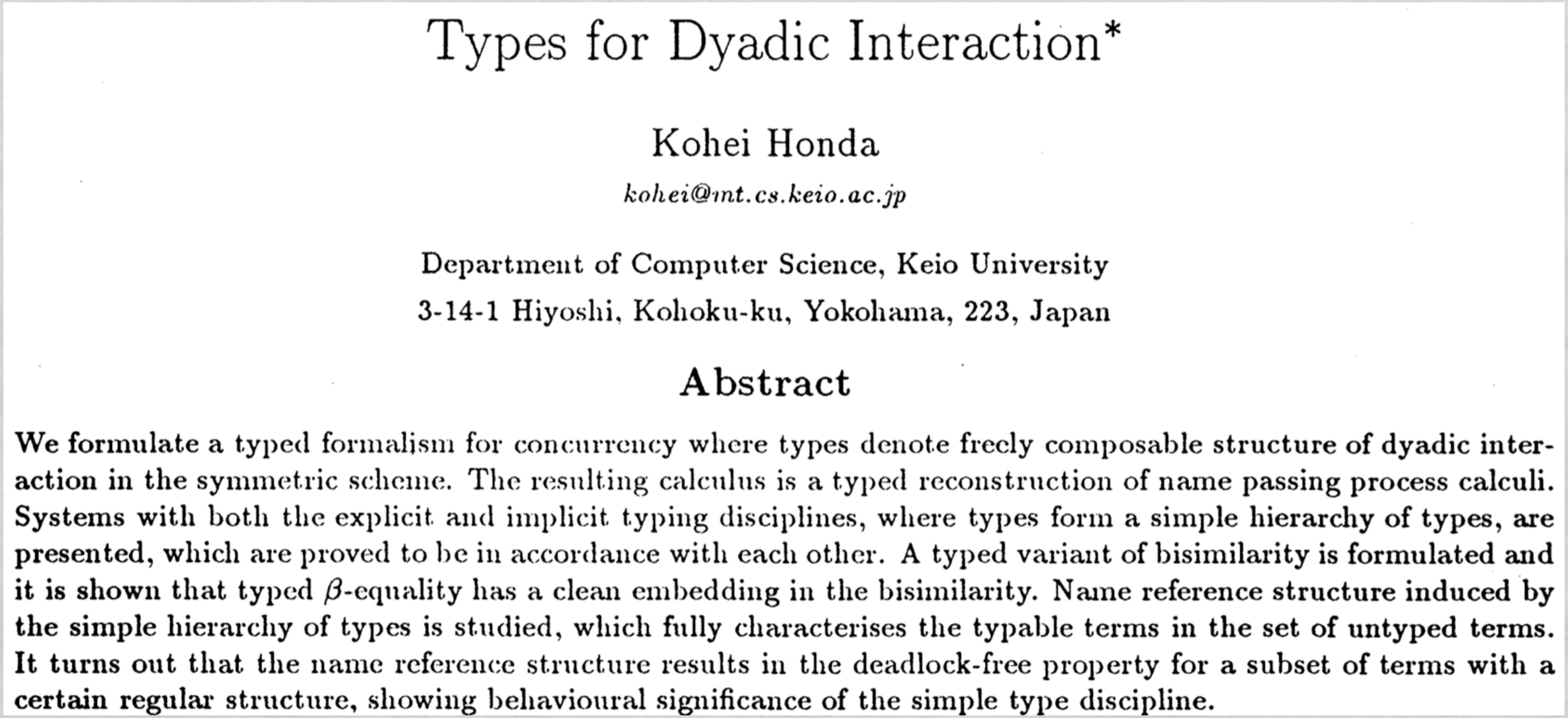
$$\frac{\Gamma;\Delta_1 \vdash y{:}T \quad \Gamma; \Delta_2, x^{p}{:}S \vdash P}{\Gamma; \Delta_1, x^{p}{:}!T.S, \Delta_2 \vdash x^{p}[y].P} \qquad \frac{\Gamma, x{:}T; \Delta \vdash P}{\Gamma; \Delta, x{:}{*T} \vdash P} \qquad \Gamma; \overline{\mathsf{end}} \vdash 0$$

$$\Gamma; x{:}U \vdash x{:}U$$

$$\frac{\Gamma; x^{p}{:}S, y{:}T \vdash P}{\Gamma; x^{p}{:}?T.S \ \vdash x^{p}(y).P} \qquad \frac{\Gamma, x{:}T; \Delta \vdash P}{\Gamma; \Delta \vdash (\mathsf{new}\ x)P} \qquad \Gamma, x{:}U; \cdot \vdash x{:}{*U}$$

## Curry Howard for Process Types?

# Session Types [Honda93]



Types for Dyadic Interaction*

Kohei Honda

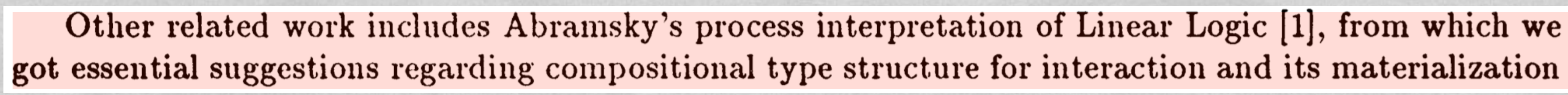
*kohei@mt.cs.keio.ac.jp*

Department of Computer Science, Keio University

3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223, Japan

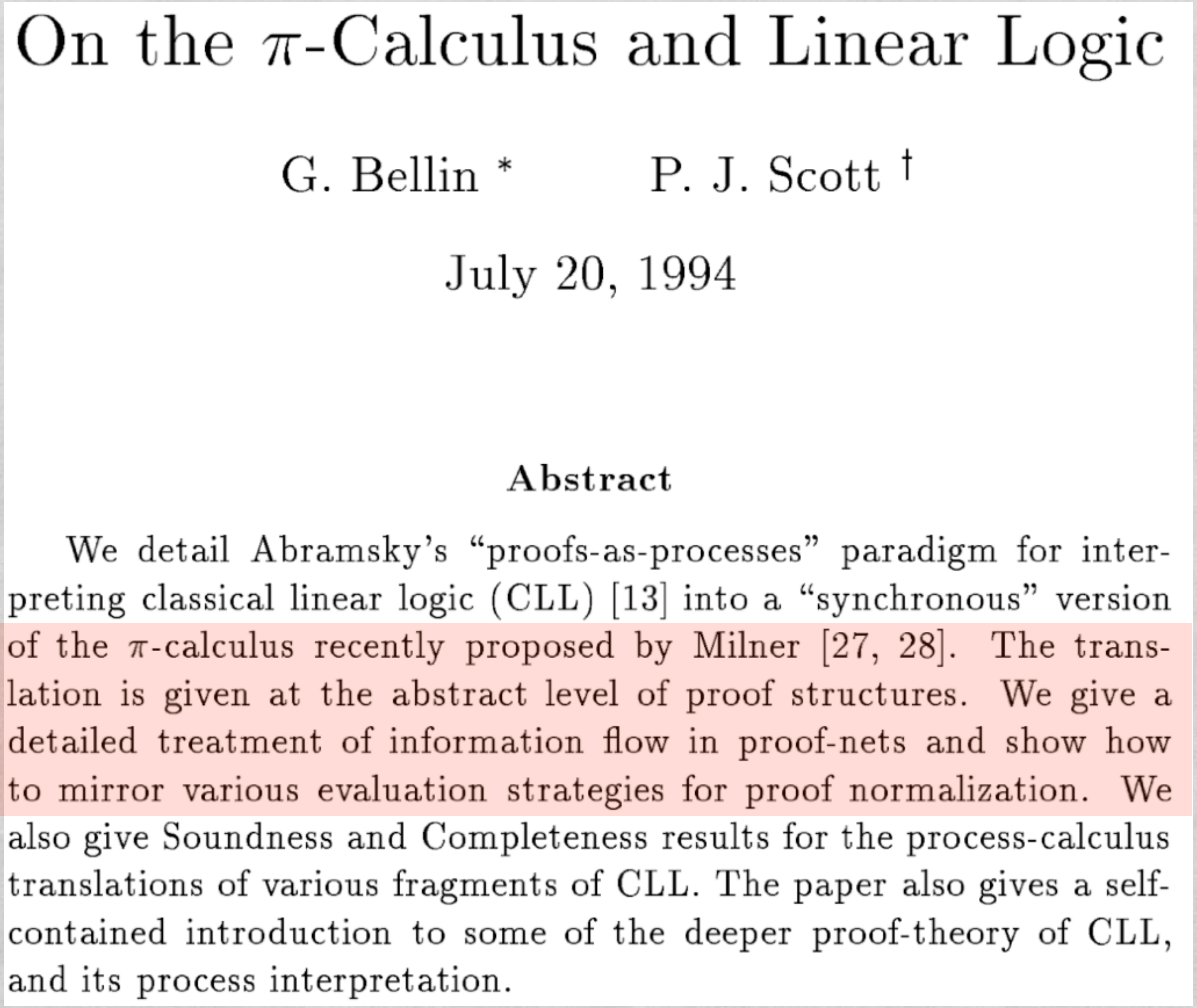
**Abstract**

We formulate a typed formalism for concurrency where types denote freely composable structure of dyadic interaction in the symmetric scheme. The resulting calculus is a typed reconstruction of name passing process calculi. Systems with both the explicit and implicit typing disciplines, where types form a simple hierarchy of types, are presented, which are proved to be in accordance with each other. A typed variant of bisimilarity is formulated and it is shown that typed $\beta$-equality has a clean embedding in the bisimilarity. Name reference structure induced by the simple hierarchy of types is studied, which fully characterises the typable terms in the set of untyped terms. It turns out that the name reference structure results in the deadlock-free property for a subset of terms with a certain regular structure, showing behavioural significance of the simple type discipline.



Other related work includes Abramsky's process interpretation of Linear Logic [1], from which we got essential suggestions regarding compositional type structure for interaction and its materialization

# Computational Interpretations of LL



On the $\pi$-Calculus and Linear Logic

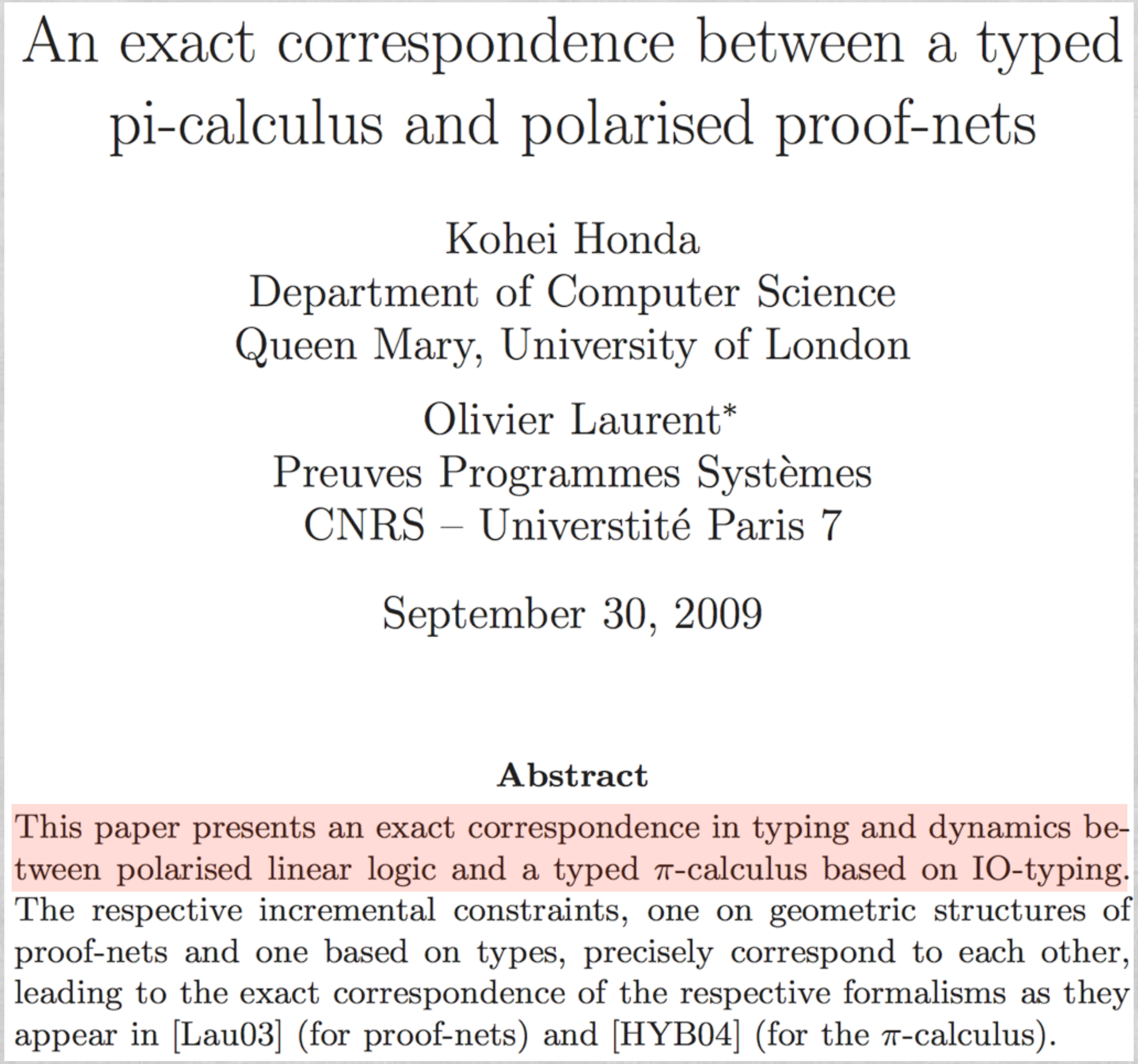
G. Bellin [*]  P. J. Scott [†]

July 20, 1994

**Abstract**

We detail Abramsky's "proofs-as-processes" paradigm for interpreting classical linear logic (CLL) [13] into a "synchronous" version of the $\pi$-calculus recently proposed by Milner [27, 28]. The translation is given at the abstract level of proof structures. We give a detailed treatment of information flow in proof-nets and show how to mirror various evaluation strategies for proof normalization. We also give Soundness and Completeness results for the process-calculus translations of various fragments of CLL. The paper also gives a self-contained introduction to some of the deeper proof-theory of CLL, and its process interpretation.

# Computational Interpretations of LL



An exact correspondence between a typed pi-calculus and polarised proof-nets

Kohei Honda
Department of Computer Science
Queen Mary, University of London

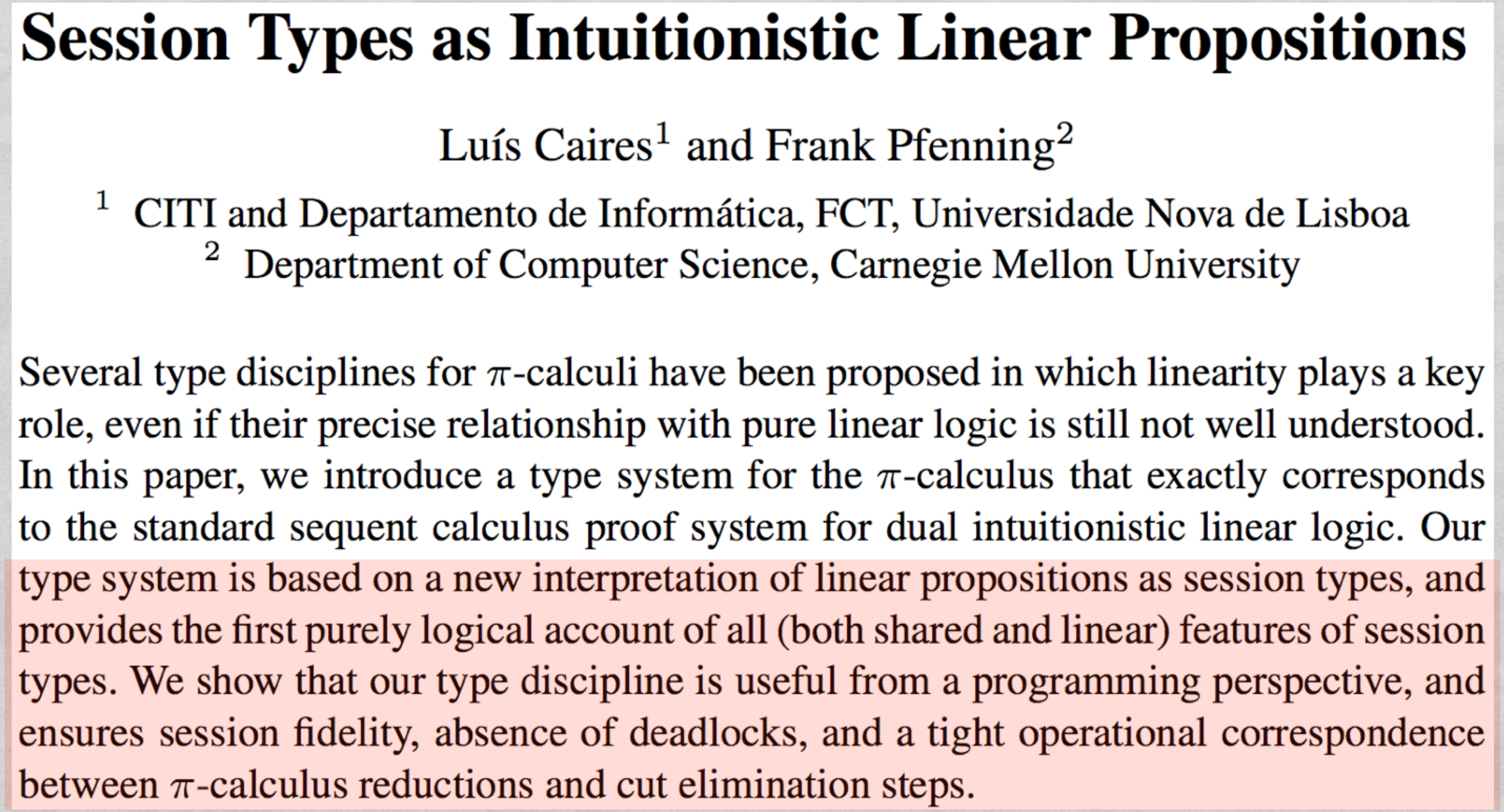
Olivier Laurent*
Preuves Programmes Systèmes
CNRS – Universtité Paris 7

September 30, 2009

**Abstract**

This paper presents an exact correspondence in typing and dynamics between polarised linear logic and a typed $\pi$-calculus based on IO-typing. The respective incremental constraints, one on geometric structures of proof-nets and one based on types, precisely correspond to each other, leading to the exact correspondence of the respective formalisms as they appear in [Lau03] (for proof-nets) and [HYB04] (for the $\pi$-calculus).

## Session Types [CairesPfenning10, ToninhoCP11-16]



**Session Types as Intuitionistic Linear Propositions**

Luís Caires[1] and Frank Pfenning[2]

[1] CITI and Departamento de Informática, FCT, Universidade Nova de Lisboa
[2] Department of Computer Science, Carnegie Mellon University

Several type disciplines for $\pi$-calculi have been proposed in which linearity plays a key role, even if their precise relationship with pure linear logic is still not well understood. In this paper, we introduce a type system for the $\pi$-calculus that exactly corresponds to the standard sequent calculus proof system for dual intuitionistic linear logic. Our type system is based on a new interpretation of linear propositions as session types, and provides the first purely logical account of all (both shared and linear) features of session types. We show that our type discipline is useful from a programming perspective, and ensures session fidelity, absence of deadlocks, and a tight operational correspondence between $\pi$-calculus reductions and cut elimination steps.

# Session Types as Linear Propositions

$S,U ::=$

| | | |
|---|---|---|
| \| | $U \multimap S$ | (input) |
| \| | $U \otimes S$ | (output) |
| \| | $S \oplus S$ | (choice) |
| \| | $S \& S$ | (offer) |
| \| | $!U$ | (shared replication) |
| \| | $1$ | (end) |

Duality on session types, the key insight of [H93], is captured by duality (or left-write symmetry) at the logical level.

## Linear Sequent Calculus [Andreolli92]

$\Gamma; \Delta \vdash A$

- $\Delta$ linear context (multiset)
- $\Gamma$ cartesian context (set)

$$\Gamma; A \vdash A \qquad \frac{\Gamma; \Delta_1 \vdash A \quad \Gamma; \Delta_2, A \vdash C}{\Gamma; \Delta_1, \Delta_2 \vdash C}$$

$$\frac{\Gamma; \Delta_1 \vdash A \quad \Gamma; \Delta_2, B \vdash C}{\Gamma; \Delta_1, \Delta_2, A \multimap B \vdash C} \qquad \frac{\Gamma; \Delta, A \vdash B}{\Gamma; \Delta \vdash A \multimap B} \qquad \Gamma; - \vdash 1$$

$$\frac{\Gamma; \Delta_1 \vdash A \quad \Gamma; \Delta_2 \vdash B}{\Gamma; \Delta_1, \Delta_2 \vdash A \otimes B} \qquad \frac{\Gamma; \Delta, A, B \vdash C}{\Gamma; \Delta, A \otimes B \vdash C} \qquad \frac{\Gamma; \Delta \vdash C}{\Gamma; \Delta, 1 \vdash C}$$

Sequent calculus presentation of DILL [BarberPlotkin91]

# Linear Propositions as Session Types

$$\Gamma; x{:}A \vdash [x \leftrightarrow y] :: y{:}A$$

$$\frac{\Gamma;\Delta_1 \vdash Q :: x{:}A \quad \Gamma;\Delta_2, x{:}A \vdash P :: C}{\Gamma; \Delta_1, \Delta_2 \vdash (\mathsf{new}\ x)(Q \mid P) :: C}$$

$$\frac{\Gamma; \Delta \vdash P :: C}{\Gamma; \Delta, y{:}1 \vdash P :: C}$$

$$\Gamma; \vdash 0 :: y{:}1$$

$$\frac{\Gamma; \Delta_1 \vdash Q :: y{:}A \quad \Gamma;\Delta_2 \vdash P :: x{:}B}{\Gamma; \Delta_1, \Delta_2 \vdash \overline{x}(y).(Q \mid P) :: x{:}A \otimes B}$$

$$\frac{\Gamma; \Delta, z{:}A, x{:}B \vdash P :: C}{\Gamma; \Delta, x{:}A \otimes B \vdash x(z).P :: C}$$

$$\frac{\Gamma; \Delta_1 \vdash Q :: y{:}A \quad \Gamma; \Delta_2, x{:}B \vdash P :: C}{\Gamma; \Delta_1, \Delta_2, x{:}A \multimap B \vdash \overline{x}(y).(Q \mid P) :: C}$$

$$\frac{\Gamma; \Delta, z{:}A \vdash P :: x{:}B}{\Gamma; \Delta \vdash x(z).P :: x{:}A \multimap B}$$

# Linear Propositions as Session Types

- Typing judgement

$$x_1{:}A_1, \ldots, x_n{:}A_n \vdash P :: y{:}C$$

- Intuition: judgement states a "rely-guarantee" property:

Whenever composed with any processes offering a session of type $A_i$ at $x_n$, process *P* will offer a session of type $C$ at $y$

$$\frac{\Gamma;\Delta_1 \vdash Q :: x{:}A \quad \Gamma;\Delta_2, x{:}A \vdash P :: C}{\Gamma;\, \Delta_1, \Delta_2 \vdash (\mathsf{new}\ x)(Q \mid P) :: C}$$

Typing ensures fidelity and global progress (cut-elimination)

# Movie Server Session

$SrvBody(s) \triangleq s.\mathtt{case}(\ s(title).s(card).\overline{s}(\text{movie}).\mathbf{0};$
$\qquad s(title).\overline{s}(trailer).\mathbf{0})$

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

$System \triangleq (\mathtt{new}\ s)(\ SrvBody(s) \mid Alice(s)\ )$

$ServerProto \triangleq (\text{Name} \multimap \text{CardN} \multimap (\text{MP4}\otimes\mathbf{1}))\&(\text{Name} \multimap (\text{MP4}\otimes\mathbf{1})$

$- ; - \vdash SrvBody(s) :: s{:}ServerProto$

$- ; s{:}ServerProto \vdash BClntBody(s) :: -{:}\mathbf{1}$

$- ; - \vdash System :: -{:}\mathbf{1}$

## Shared Movie Server

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

$SAlice(s) \triangleq \overline{srv}(s).s.\mathtt{inr};\overline{s}(\text{“}solaris\text{”}).s(preview).\mathbf{0}$

$SBob(s) \triangleq \overline{srv}(s).s.\mathtt{inl};\overline{s}(\text{“}inception\text{”}).\overline{s}(\text{“8888”}).s(movie).\mathbf{0}$

$SSystem \triangleq (\mathtt{new}\ srv)(\ Movies(srv))\ |\ SAlice(srv)\ |\ SBob(srv)\ )$

$- ; - \vdash Movies(srv) :: srv{:}!ServerProto$

$srv{:}ServerProto ; - \vdash SAlice(srv) :: -{:}\mathbf{1}$

$srv{:}ServerProto ; - \vdash SBob(srv) \quad :: -{:}\mathbf{1}$

$- ; - \vdash SSystem :: -{:}\mathbf{1}$

# Send and Receive

$$\dfrac{\dfrac{\Gamma; \Delta_1 \vdash Q :: y{:}A \quad \Gamma;\Delta_2 \vdash P :: x{:}B}{\Gamma; \Delta_1, \Delta_2 \vdash \bar{x}(y).(Q \mid P) :: x{:}A \otimes B} \qquad \dfrac{\Gamma; \Delta_3, z{:}A, x{:}B \vdash R :: C}{\Gamma; \Delta_3, x{:}A \otimes B \vdash x(z).R :: C}}{\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (\mathsf{new}\ x)(\bar{x}(y).(Q \mid P) \mid x(z).R) :: C}$$

$$\rightarrow$$

# Send and Receive

$$\dfrac{\dfrac{\Gamma; \Delta_1 \vdash Q :: y{:}A \quad \Gamma;\Delta_2 \vdash P :: x{:}B}{\Gamma; \Delta_1, \Delta_2 \vdash \bar{x}(y).(Q \mid P) :: x{:}A \otimes B} \qquad \dfrac{\Gamma; \Delta_3, y{:}A, x{:}B \vdash R :: C}{\Gamma; \Delta_3, x{:}A \otimes B \vdash x(y).R :: C}}{\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (\mathsf{new}\ x)(\bar{x}(y).(Q \mid P) \mid x(y).R) :: C}$$

$$\rightarrow$$

# Send and Receive

$$\dfrac{\dfrac{\Gamma; \Delta_1 \vdash Q :: y{:}A \qquad \Gamma;\Delta_2 \vdash P :: x{:}B}{\Gamma; \Delta_1, \Delta_2 \vdash \bar{x}(y).(Q \mid P) :: x{:}A \otimes B} \qquad \dfrac{\Gamma; \Delta_3, y{:}A, x{:}B \vdash R :: C}{\Gamma; \Delta_3, x{:}A \otimes B \vdash x(y).R :: C}}{\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (\mathsf{new}\ x)(\bar{x}(y).(Q \mid P) \mid x(y).R) :: C}$$

$$\to$$

$$\dfrac{\Gamma;\Delta_2 \vdash P :: x{:}B \qquad \dfrac{\Gamma; \Delta_1 \vdash Q :: y{:}A \qquad \Gamma; \Delta_3, y{:}A, x{:}B \vdash R :: C}{\Gamma; \Delta_1, \Delta_3, x{:}B \vdash (\mathsf{new}\ y)(Q \mid R) :: C}}{\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (\mathsf{new}\ x)(P \mid (\mathsf{new}\ y)(Q \mid R)) :: C}$$

$\mathsf{Tcut}[x](\mathsf{TR}\otimes[y](d_1, d_2), \mathsf{TL}\otimes[y](d_3)) \to \mathsf{Tcut}[x](d_2, \mathsf{Tcut}[y](d_1, d_3))$

# Send and Receive

$$\dfrac{\dfrac{\Gamma; \Delta_1, y{:}A \vdash P :: x{:}B}{\Gamma; \Delta_1 \vdash x(y).P :: x{:}A \multimap B} \qquad \dfrac{\Gamma; \Delta_2 \vdash Q :: y{:}A \quad \Gamma; \Delta_3, x{:}B \vdash R :: C}{\Gamma; \Delta_2, \Delta_3, x{:}A \multimap B \vdash \bar{x}(y).(Q \mid R) :: C}}{\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (\text{new } x)(x(y).P \mid \bar{x}(y).(Q \mid R)) :: C}$$

$$\rightarrow$$

## Send and Receive

$$\dfrac{\dfrac{\Gamma; \Delta_1, y{:}A \vdash P :: x{:}B}{\Gamma; \Delta_1 \vdash x(y).P :: x{:}A \multimap B} \qquad \dfrac{\Gamma; \Delta_2 \vdash Q :: y{:}A \quad \Gamma; \Delta_3, x{:}B \vdash R :: C}{\Gamma; \Delta_2, \Delta_3, x{:}A \multimap B \vdash \bar{x}(y).(Q \mid R) :: C}}{\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (\text{new}\ x)(x(y).P \mid \bar{x}(y).(Q \mid R)) :: C}$$

$$\rightarrow$$

$$\dfrac{\dfrac{\Gamma; \Delta_2 \vdash Q :: y{:}A \qquad \Gamma; \Delta_1, y{:}A \vdash P :: x{:}B}{\Gamma; \Delta_2, \Delta_1, x{:}B \vdash (\text{new}\ y)(Q \mid P) :: x{:}B} \qquad \Gamma; \Delta_3, x{:}B \vdash R :: C}{\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash (\text{new}\ x)((\text{new}\ y)(Q \mid P) \mid R) :: C}$$

$\mathsf{Tcut}[x](\mathsf{TR}\multimap[y](d_1), \mathsf{TL}\multimap[y](d_2, d_3)) \rightarrow \mathsf{Tcut}[x](\mathsf{Tcut}[y](d_2, d_1), d_3)$

## Replication and Sharing

$$\frac{\Gamma; \vdash P :: y : A}{\Gamma; \vdash\ !x(y).P :: x : !A}$$

$$\frac{\Gamma, x{:}A; \Delta, y{:}A \vdash P :: C}{\Gamma, x{:}A; \Delta \vdash \bar{x}(y).P :: C}$$

$$\frac{\Gamma, x{:}A; \Delta \vdash P :: C}{\Gamma; \Delta, x{:}!A \vdash P :: C}$$

## Replication and Sharing

$$\dfrac{\dfrac{\Gamma; \vdash P :: y{:}A}{\Gamma; \vdash\ !x(y).P :: x{:}!A} \qquad \dfrac{\Gamma, x{:}A; \Delta, y{:}A \vdash Q :: C}{\Gamma, x{:}A\, ; \Delta \vdash \bar{x}(y).Q :: C}}{\Gamma; \Delta \vdash (\mathsf{new}\ x)(!x(y).P \mid \bar{x}(y).Q) :: C}$$

$$\rightarrow$$

$$\dfrac{\Gamma; \vdash P :: y{:}A \qquad \dfrac{\Gamma; \vdash P :: y{:}A \quad \Gamma, x{:}A; \Delta, y{:}A \vdash Q :: C}{\Gamma; \Delta, y{:}A \vdash (\mathsf{new}\ x)(!x(y).P \mid Q) :: C}}{\Gamma; \Delta \vdash (\mathsf{new}\ y)(P \mid (\mathsf{new}\ x)(!x(y).P \mid Q)) :: C}$$

Tcut$[x]$(TR!$[y](d_1)$, TL!$[z](d_2)$) $\rightarrow$ Tcut$[x](d_1,$ Tcut!$[xy](d_1, d_2))$

# Choice and Offer

$$\frac{\Gamma; \Delta \vdash Q :: x{:}\mathrm{A} \qquad \Gamma; \Delta \vdash \mathrm{P} :: x{:}\mathrm{B}}{\Gamma; \Delta \vdash x.\texttt{case}(Q,P) :: x{:}\mathrm{A}\&\mathrm{B}}$$

$$\frac{\Gamma; \Delta, x{:}\mathrm{A} \vdash R :: \mathrm{C}}{\Gamma; \Delta, x{:}\mathrm{A}\&\mathrm{B} \vdash x.\texttt{inl};R :: \mathrm{C}}$$

$$\frac{\Gamma; \Delta \vdash \mathrm{P} :: x{:}\mathrm{B}}{\Gamma; \Delta \vdash x.\texttt{inr};\mathrm{P} :: x{:}\mathrm{A}\oplus\mathrm{B}}$$

$$\frac{\Gamma; \Delta, x{:}\mathrm{B} \vdash R :: \mathrm{C}}{\Gamma; \Delta, x{:}\mathrm{A}\&\mathrm{B} \vdash x.\texttt{inr};R :: \mathrm{C}}$$

$$\frac{\Gamma; \Delta \vdash \mathrm{P} :: x{:}\mathrm{A}}{\Gamma; \Delta \vdash x.\texttt{inl};\mathrm{P} :: x{:}\mathrm{A}\oplus\mathrm{B}}$$

$$\frac{\Gamma; \Delta, x{:}\mathrm{A} \vdash Q :: C \qquad \Gamma; \Delta, x{:}\mathrm{B} \vdash P :: C}{\Gamma; \Delta, x{:}\mathrm{A}\oplus\mathrm{B} \vdash x.\texttt{case}(Q,P) :: \mathrm{C}}$$

## Choice and Offer

$$\dfrac{\dfrac{\Gamma; \Delta_1 \vdash Q :: x{:}A \qquad \Gamma; \Delta_1 \vdash P :: x{:}B}{\Gamma; \Delta_1 \vdash x.\mathtt{case}(Q,P) :: x{:}A\&B} \qquad \dfrac{\Gamma; \Delta_2, x{:}A \vdash R :: C}{\Gamma; \Delta_2, x{:}A\&B \vdash x.\mathtt{inl};R :: C}}{\Gamma; \Delta_1, \Delta_2 \vdash (\mathsf{new}\ x)(x.\mathtt{case}(Q,P) \mid x.\mathtt{inl};R) :: C}$$

$$\rightarrow$$

$$\dfrac{\Gamma; \Delta_1 \vdash Q :: x{:}A \qquad \Gamma; \Delta_2, x{:}A \vdash R :: C}{\Gamma; \Delta_1, \Delta_2 \vdash (\mathsf{new}\ x)(Q \mid R) :: C}$$

$\mathsf{Tcut}[x](\mathsf{TR\&}(d_1, d_2), \mathsf{TL1\&}(d_3)) \rightarrow \mathsf{Tcut}[x](d_1, d_3)$

$\mathsf{Tcut}[x](\mathsf{TR\&}(d_1, d_2), \mathsf{TL2\&}(d_3)) \rightarrow \mathsf{Tcut}[x](d_2, d_3)$

## Choice and Offer (labeled sums)

$$\frac{\Gamma; \Delta \vdash P_i :: x{:}A_i}{\Gamma; \Delta \vdash x.\mathtt{case}(\mathtt{l}_i{:}P_i) :: x{:}\&\{\mathtt{l}_i{:}A_i\}} \qquad \frac{\Gamma; \Delta, x{:}A_i \vdash Q :: C}{\Gamma; \Delta, x{:}\&\{\mathtt{l}_i{:}A_i\} \vdash x.\mathtt{l}_i; Q :: C}$$

$$\frac{\Gamma; \Delta \vdash P :: x{:}A_i}{\Gamma; \Delta \vdash x.\mathtt{l}_i; P :: x{:}\oplus\{\mathtt{l}_i{:}A_i\}} \qquad \frac{\Gamma; \Delta, x{:}A_i \vdash P_i :: C}{\Gamma; \Delta, x{:}\oplus\{\mathtt{l}_i{:}A_i\} \vdash x.\mathtt{case}(\mathtt{l}_i{:}P_i) :: C}$$

$$\&\{\mathtt{l}_i{:}A_i\} \triangleq A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_n$$
$$\oplus\{\mathtt{l}_i{:}A_i\} \triangleq A_1 \oplus A_2 \oplus \ldots \oplus A_n$$

## Copycat Forwarder

$$\frac{\Gamma; x{:}A \vdash [x\leftrightarrow y] :: y{:}A \qquad \Gamma; \Delta, y{:}A \vdash P :: C}{\Gamma; \Delta, x{:}A \vdash (\mathsf{new}\ x)(\,[x\leftrightarrow y] \mid P) :: C}$$

$$\rightarrow$$

$$\Gamma; \Delta, x{:}A \vdash P\{x/y\} :: C$$

$\mathsf{Tcut}[x](\mathsf{TA}[xy], d)) \rightarrow d\{x/y\}$

The axiom forwarder already appears in [Abramsky01], but used very differently.

# Some Admissible Rules in DILL

$$\frac{\Gamma; \Delta_1 \vdash P :: 1 \quad \Gamma; \Delta_2 \vdash Q :: C}{\Gamma; \Delta_1, \Delta_2 \vdash P \mid Q :: C}$$

cf. the so-called mix rule (independent composition)

$$\Gamma; 1 \vdash \mathbf{0} :: 1$$

cf. empty
(replacing T1R and T1L)
Exactly as in [GH05] Tend

$$\frac{\Gamma;\Delta \vdash P :: x{:}B}{\Gamma; y{:}A, \Delta \vdash x[y].P :: x{:}A \otimes B}$$

$$x[y].P \triangleq \overline{x}(z).([z \leftrightarrow y] \| P)$$

cf. internal mobility
translation [Boreale98]

## Duality in DILL

$$S ::= 1 \mid U \otimes S \mid U \multimap S \mid S \oplus S \mid S \& S$$

$$\begin{aligned}
\overline{U \otimes S} &= U \multimap \overline{S} \\
\overline{U \multimap S} &= U \otimes \overline{S} \\
\overline{S \oplus S} &= \overline{S} \& \overline{S} \\
\overline{S \& S} &= \overline{S} \oplus \overline{S} \\
\overline{1} &= 1
\end{aligned}
\qquad\qquad
\overline{\overline{S}} = S$$

**Theorem**. $\Gamma; \Delta \vdash P :: x{:}U$ *if and only if* $\Gamma; \Delta, x{:}\overline{U} \vdash P :: -{:}1$

Duality on session types captured by left-right symmetry

# Proofs = Processes

| | | | |
|---|---|---|---|
| $P$ | $::=$ | $0$ | (inaction) |
| | $\mid$ | $[x \leftrightarrow y]$ | (linear forwarder) |
| | $\mid$ | $(\mathtt{new}\ x)(P \mid Q)$ | (composition) |
| | $\mid$ | $x(y).P$ | (input) |
| | $\mid$ | $\bar{x}(y).P$ | (output) |
| | $\mid$ | $!x(y).P$ | (replicated server) |
| | $\mid$ | $x.\mathtt{case}(P,Q)$ | (offer) |
| | $\mid$ | $x.\mathtt{inl};Q$ | (choose left) |
| | $\mid$ | $x.\mathtt{inr};Q$ | (choose right) |

# Proof Conversions = Process Identities

Structural Conversions ( $\equiv$ )

Identify structurally identical proofs (e.g, commute cuts, expose redexes)

Correspond to standard structural congruences ( $\equiv$ )

$(\text{new}\ x)(0 \mid P) \ \equiv\ P$
$(\text{new}\ x)(P \mid (\text{new}\ y)(Q \mid R)) \equiv\ (\text{new}\ y)((\text{new}\ x)(P \mid Q) \mid R)$
$(\text{new}\ x)(P \mid (\text{new}\ y)(Q \mid R)) \equiv\ (\text{new}\ y)(Q \mid (\text{new}\ x)(P \mid R))$

## Proof Reductions = Process Reductions

Computational Conversions ($\rightarrow$)

Reduce proofs into simpler ones (e.g, decreases types)

Correspond to standard process reductions ( $\rightarrow$ )

$(\mathsf{new}\ x)(x(y).P \mid \bar{x}(y).(Q \mid R)) \rightarrow (\mathsf{new}\ x)(P \mid (\mathsf{new}\ y)(Q \mid \mathrm{R}))$

$(\mathsf{new}\ x)(x.\mathtt{case}(Q,P) \mid x.\mathtt{inl};R) \rightarrow (\mathsf{new}\ x)(Q \mid R)$

$(\mathsf{new}\ x)(!x(y).P \mid \bar{x}(y).Q) \rightarrow (\mathsf{new}\ y)(P \mid (\mathsf{new}\ x)(!x(y).P \mid Q))$

## Proof Conversions = Process Identities

- Structural Conversions ($\simeq$)

  Correspond to well known typed strong bisimilarities ( $\simeq$ )

  (new $x$)($!x(y).P$ | (new $z$)($Q$ | $R$)) $\simeq$
  (new $z$)( (new $x$)($!x(y).P$ | $Q$) | (new $x$)($!x(y).P$ | $R$))

  (new $x$)($!x(y).P$ | (new $z$)($!z(u).Q$ | $R$)) $\simeq$
  (new $z$)( $!z(u)$.(new $x$)($!x(y).P$ | $Q$) | (new $x$)($!x(y).P$ | $R$))

  (new $x$)($!x(y).P$ | $Q$) $\simeq Q$ $[\ x \notin fn(Q)\ ]$

- The **sharpened replication lemmas** of [SangiorgiWalker01]
- Yet another remarkable bridge surfaces here

# Proof Conversions = Process Identities

- Structural Conversions ( $\equiv$ )

  ( $\equiv$ ) matched by $\pi$ structural congruence ( $\equiv$ )

- Computational Conversions ($\longrightarrow$)

  ($\longrightarrow$) matched by $\pi$ reduction ( $\rightarrow$ )

- Structural Conversions ( $\simeq$ )

  ( $\simeq$ ) matched by typed $\pi$ observational equivalence ( $\equiv$ )

- All Conversions ( $\cong$ )

# Curry-Howard Correspondence

**Theorem** (*processes as proofs*) [CairesPfenning10,CPT*]

If $\Gamma; \Delta \vdash P :: C$ and $P \equiv \rightarrow \equiv Q$ then $\Gamma; \Delta \vdash P \cong \rightarrow \cong Q :: C$

**Theorem** (*proofs as processes)* [CairesPfenning10,CPT*]

If $\Gamma; \Delta \vdash P \rightarrow Q :: C$ then $P \rightarrow Q$

If $\Gamma; \Delta \vdash P \equiv Q :: C$ then $P \equiv Q$

If $\Gamma; \Delta \vdash P \simeq Q :: C$ then $P \simeq Q$

# Curry-Howard Correspondence

**Theorem** (*progress*) [CairesPfenning10,CPT*]

$live(P) \triangleq P \not\equiv 0$

If $-; - \vdash P :: -:1$ and $live(P)$ then $P \rightarrow Q$

## From Theorems to Code

- Every **provable** sequent $\Gamma; \Delta \vdash C$ "is" a process $\Gamma; \Delta \vdash P :: C$
- We may "automatically" produce interface adapters for every linear logic theorem, e.g., $x{:}A \vdash P :: y{:}B$ is a morphism A→B
- Examples (try to figure out what the process is )

  $x{:}X \otimes Y \vdash y{:}Y \otimes X$

  $x{:}X \multimap (Y \& Z) \vdash y{:}\ (X \multimap Y) \ \&\ (X \multimap Z)$

- Generally [ESOP'12], an isomorphism A $\leftrightharpoons$ B is process pair ($P$, $Q$) such that $x{:}A \vdash P :: y{:}B$ *and* $y{:}B \vdash Q :: x{:}A$ *and*

  $x{:}A \vdash (\mathsf{new}\ y)(P | Q\{z/x\}) \approx [x \leftrightarrow z] :: z{:}A$

  $y{:}B \vdash (\mathsf{new}\ z)(Q | P\{z/y\}) \approx [y \leftrightarrow z] :: z{:}B$

# Movie Server Session

$$\mathit{SrvBody}(s) \triangleq s.\mathtt{case}(\ s(\mathit{title}).s(\mathit{card}).\overline{s}(\mathrm{movie}).\mathbf{0};\\ s(\mathit{title}).\overline{s}(\mathit{trailer}).\mathbf{0})$$

$$\mathit{Alice}(s) \triangleq s.\mathtt{inr};\overline{s}(\text{“}\mathit{solaris}\text{”}).s(\mathit{preview}).\mathbf{0}$$

$$\mathit{System} \triangleq (\mathtt{new}\ s)(\ \mathit{SrvBody}(s) \mid \mathit{Alice}(s)\ )$$

$$\mathrm{ServerProto} \triangleq (\mathrm{Name} \multimap \mathrm{CardN} \multimap (\mathrm{MP4}\otimes\mathbf{1}))\&(\mathrm{Name} \multimap (\mathrm{MP4}\otimes\mathbf{1}))$$

$$\text{-}\ ;\ \text{-} \vdash \mathit{SrvBody}(s) :: s{:}\mathit{ServerProto}$$

$$\text{-}\ ;\ s{:}\mathit{ServerProto} \vdash \mathit{BClntBody}(s) :: \text{-}{:}\mathbf{1}$$

$$\text{-}\ ;\ \text{-} \vdash \mathit{System} :: \text{-}{:}\mathbf{1}$$

# Movie Server Session

$- ; - \vdash (\mathsf{new}\ s)(\ \mathit{SrvBody}(s) \mid \mathit{Alice}(s)\ ) :: -{:}\mathbf{1}$

$\rightarrow$ $\mathsf{Tcut}[s](\mathsf{TR\&}(d_1, d_2), \mathsf{TL2\&}(d_3)) \rightarrow \mathsf{Tcut}[s](d_1, d_2)$

$- ; - \vdash (\mathsf{new}\ s)(\ s(\mathit{title}).\overline{s}(\mathit{trailer}).\mathbf{0} \mid \overline{s}(\text{“}\mathit{solaris}\text{”}).s(\mathit{preview}).\mathbf{0}\ ) :: -{:}\mathbf{1}$

$\rightarrow$ $\mathsf{Tcut}[s](\mathsf{TR}{\multimap}(d_1), \mathsf{TL}{\multimap}(d_2, d_3)) \rightarrow \mathsf{Tcut}[s](\mathsf{Tcut}(d_2, d_1), d_3)$

$- ; - \vdash (\mathsf{new}\ s)(\ \overline{s}(\mathit{trailer}).\mathbf{0} \mid s(\mathit{preview}).\mathbf{0}\ ) :: -{:}\mathbf{1}$

$\rightarrow$ $\mathsf{Tcut}[s](\mathsf{TR}{\otimes}(d_1, d_2), \mathsf{TL}{\otimes}(d_3)) \rightarrow \mathsf{Tcut}[s](d_1, \mathsf{Tcut}(d_2, d_3))$

$- ; - \vdash (\mathsf{new}\ s)(\ \mathbf{0} \mid \mathbf{0}\ ) :: -{:}\mathbf{1}$

$\equiv$ $\mathsf{Tcut}[s](\mathsf{TR1}, \mathsf{TL1}(\mathsf{TR1})) \equiv \mathsf{TR1}$

$- ; - \vdash \mathbf{0} :: -{:}\mathbf{1}$

# Replication and Sharing

$$\frac{\Gamma; \vdash P :: y : A}{\Gamma; \vdash\ !x(y).P :: x : !A}$$

$$\Gamma; x{:}A \vdash [x\leftrightarrow y] :: y{:}A$$

$$\Gamma; \vdash 0 :: y{:}1$$

$$\frac{\Gamma, x{:}A; \Delta, y{:}A \vdash P :: C}{\Gamma, x{:}A; \Delta \vdash \bar{x}(y).P :: C}$$

$$\frac{\Gamma, x{:}A; \Delta \vdash P :: C}{\Gamma; \Delta, x{:}!A \vdash P :: C}$$

$$\frac{\Gamma; \vdash P :: y{:}A \qquad \Gamma, x{:}A; \Delta \vdash Q :: C}{\Gamma; \Delta \vdash (\mathsf{new}\ x)(!x(y).P \mid Q) :: C}$$

Key idea of DILL [BarberPlotkin91]: postponing of contraction and weakening (“fat axioms”).

# Replication and Sharing

$$\dfrac{\dfrac{\Gamma; \vdash P :: y{:}A}{\Gamma; \vdash\ !x(y).P :: x{:}!A} \qquad \dfrac{\Gamma, x{:}A; \Delta \vdash Q :: C}{\Gamma; x{:}!A, \Delta \vdash Q :: C}}{\Gamma; \Delta \vdash (\mathsf{new}\ x)(!x(y).P \mid Q) :: C}$$

$$\equiv$$

$$\dfrac{\Gamma; \ \vdash P :: y{:}A \qquad \Gamma, x{:}A; \Delta \vdash Q :: C}{\Gamma; \Delta \ \vdash (\mathsf{new}\ x)(!x(y).P \mid Q) :: C}$$

$\mathsf{Tcut}[x](\mathsf{TR!}(d_1), \mathsf{TL!}[y](d_2)) \rightarrow \mathsf{Tcut!}[xy](d_1,d_2)$

# Replication and Sharing

$$\dfrac{\Gamma;\ \vdash P :: y{:}\mathrm{A} \qquad \dfrac{\Gamma, x{:}\mathrm{A}; \Delta, y{:}\mathrm{A} \vdash Q :: \mathrm{C}}{\Gamma, x{:}\mathrm{A}\,; \Delta \vdash \bar{x}(y).Q :: \mathrm{C}}}{\Gamma; \Delta \vdash (\mathsf{new}\ x)(!x(y).P \mid \bar{x}(y).Q) :: \mathrm{C}}$$

$$\rightarrow$$

$$\dfrac{\Gamma;\ \vdash P :: y{:}\mathrm{A} \qquad \dfrac{\Gamma;\ \vdash P :: y{:}\mathrm{A} \qquad \Gamma, x{:}\mathrm{A}; \Delta, y{:}\mathrm{A} \vdash Q :: \mathrm{C}}{\Gamma; \Delta, y{:}\mathrm{A} \vdash (\mathsf{new}\ x)(!x(y).P \mid Q) :: \mathrm{C}}}{\Gamma; \Delta \vdash (\mathsf{new}\ y)(P \mid (\mathsf{new}\ x)(!x(y).P \mid Q)) :: \mathrm{C}}$$

$\mathsf{Tcut!}[xy](d_1, \mathsf{Tcopy}[xy](d_2)) \rightarrow \mathsf{Tcut}[y](d_1, \mathsf{Tcut!}[xy](d_1, d_2))$

## Shared Movie Server

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

$SAlice(s) \triangleq \overline{srv}(s).s.\mathtt{inr};\overline{s}(\text{“}solaris\text{”}).s(preview).\mathbf{0}$

$SBob(s) \triangleq \overline{srv}(s).s.\mathtt{inl};\overline{s}(\text{“}inception\text{”}).\overline{s}(\text{“8888”}).s(movie).\mathbf{0}$

$SSystem \triangleq (\mathtt{new}\ srv)(\ Movies(srv)) \mid SAlice(srv) \mid SBob(srv)\ )$

$- ; - \vdash Movies(srv) :: srv{:}!ServerProto$

$srv{:}ServerProto ; - \vdash SAlice(srv) :: -{:}\mathbf{1}$

$srv{:}ServerProto ; - \vdash SBob(srv) \quad :: -{:}\mathbf{1}$

$- ; - \vdash SSystem :: -{:}\mathbf{1}$

# Shared Movie Server

- ; - ⊢ (new $srv$)( $Mov(srv)$ | $SA(srv)$ | $SB(srv)$ )

$\cong$ sharpened replication lemma (distribution of ! over | )

- ; - ⊢ (new $srv$)($Mov(srv)$ | $SA(srv)$) | (new $srv$)($Mov(srv)$ | $SB(srv)$)

$\rightarrow \equiv$ Tcut(TR!,TL!) followed by Tcut / Tcut! assoc

- ; - ⊢ . . . . . . . . (new $srv$)($Mov(srv)$ | (new $s$)($SrvBody(s)$ | $Bob(s)$))

$\cong$ sharpened replication lemma (distribution of ! over | )

- ; - ⊢ (new $srv$)( $Mov(srv)$ | $SA(srv)$ | (new $s$)($SrvBody(s)$ | $Bob(s)$))

$\rightarrow^*$

- ; - ⊢ (new $srv$)( $Mov(srv)$ | $\mathbf{0}$) $\cong \mathbf{0}$

## DILL and Locality

$$\frac{\Gamma; \vdash P :: y : A}{\Gamma; \vdash\ !x(y).P :: x : !A}$$

$$\frac{\Gamma, x{:}A; \Delta \vdash P :: C}{\Gamma; \Delta, x{:}!A \vdash P :: C} \qquad \frac{\Gamma, x{:}A; \Delta, y{:}A \vdash P :: C}{\Gamma, x{:}A; \Delta \vdash \overline{x}(y).P :: C}$$

- !A type always offered at positive polarity for server offer
- !A type always used at negative polarity for server invocation
- So a process such as $a(x).!x(y).P$ is **not typable in DILL**
- DILL enforces **locality** on shared receptive names

  ( Of course, linear sessions may still output receptive names )

## Dual Shared Types: !A and ?A

- !A

  Type for a shared channel server name that can persistently accept requests for a fresh session of type A.

- ?A

  Type for a channel name that can request creation of a fresh session of type A by communicating to a channel of type !A.

- In [GH05] such (shared) names can be freely aliased at output (invocation) and input (acceptance) modes.

However, this is not allowed in logical based disciplines.

# Dual Shared Types: !A and ?A

- Type for session that receives a channel to which server invocations of type A can be sent, and continues as B:

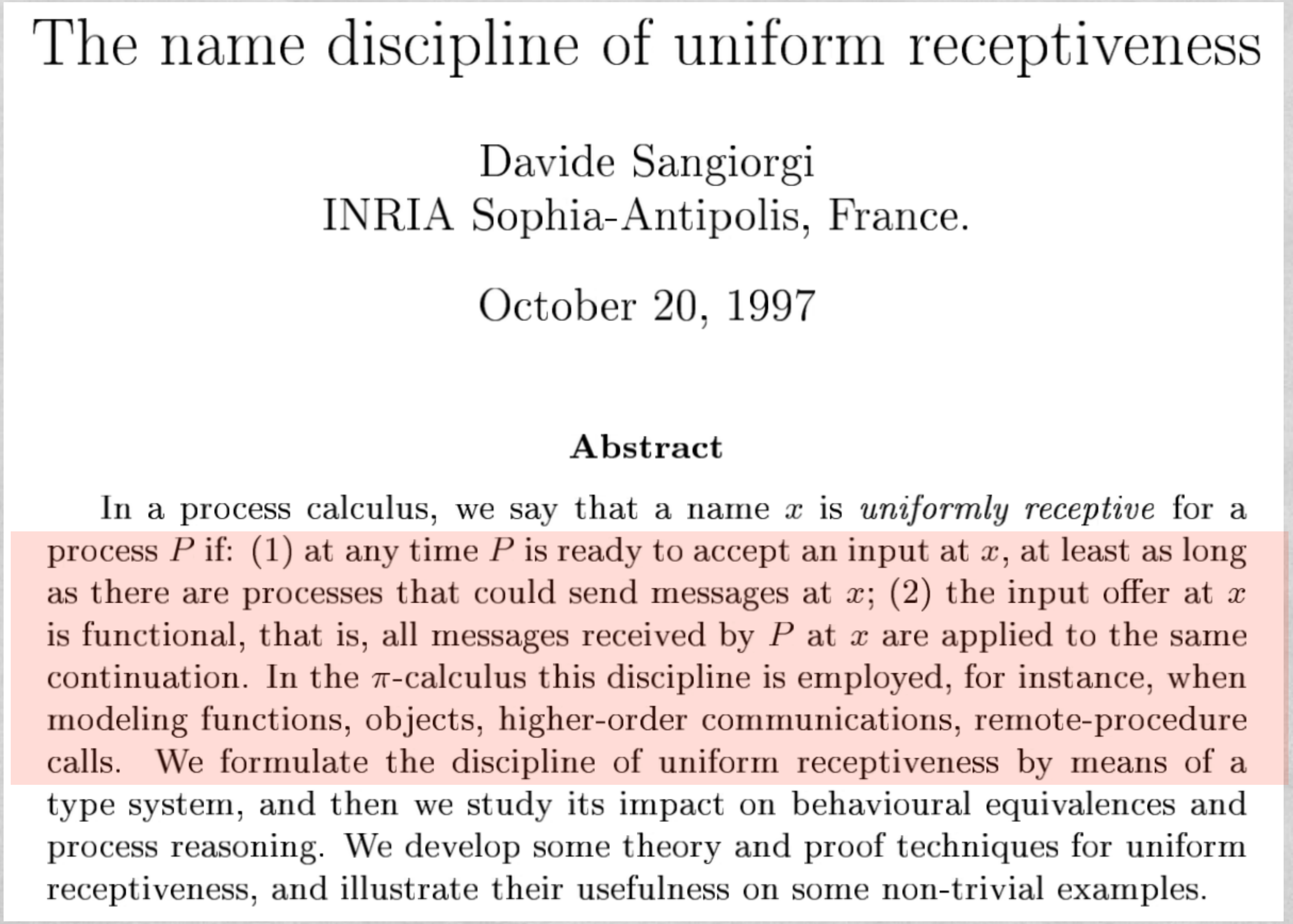
  $!A \multimap B$

- Type for session that receives a channel from which server invocations of type A can be received, and continues as B:

  $?A \multimap B$ (not expressible in DILL)

- In traditional session types [GH05], types !A and ?A get amalgamated into a unique, unpolarised, shared type [A]
- [GH05] does not enforce locality or uniform receptiveness, in the sense of [Sangiorgi97] (no non-deterministic behaviour)

# uniform receptiveness [Sangiorgi97]



## The name discipline of uniform receptiveness

Davide Sangiorgi
INRIA Sophia-Antipolis, France.

October 20, 1997

**Abstract**

In a process calculus, we say that a name $x$ is *uniformly receptive* for a process $P$ if: (1) at any time $P$ is ready to accept an input at $x$, at least as long as there are processes that could send messages at $x$; (2) the input offer at $x$ is functional, that is, all messages received by $P$ at $x$ are applied to the same continuation. In the $\pi$-calculus this discipline is employed, for instance, when modeling functions, objects, higher-order communications, remote-procedure calls. We formulate the discipline of uniform receptiveness by means of a type system, and then we study its impact on behavioural equivalences and process reasoning. We develop some theory and proof techniques for uniform receptiveness, and illustrate their usefulness on some non-trivial examples.

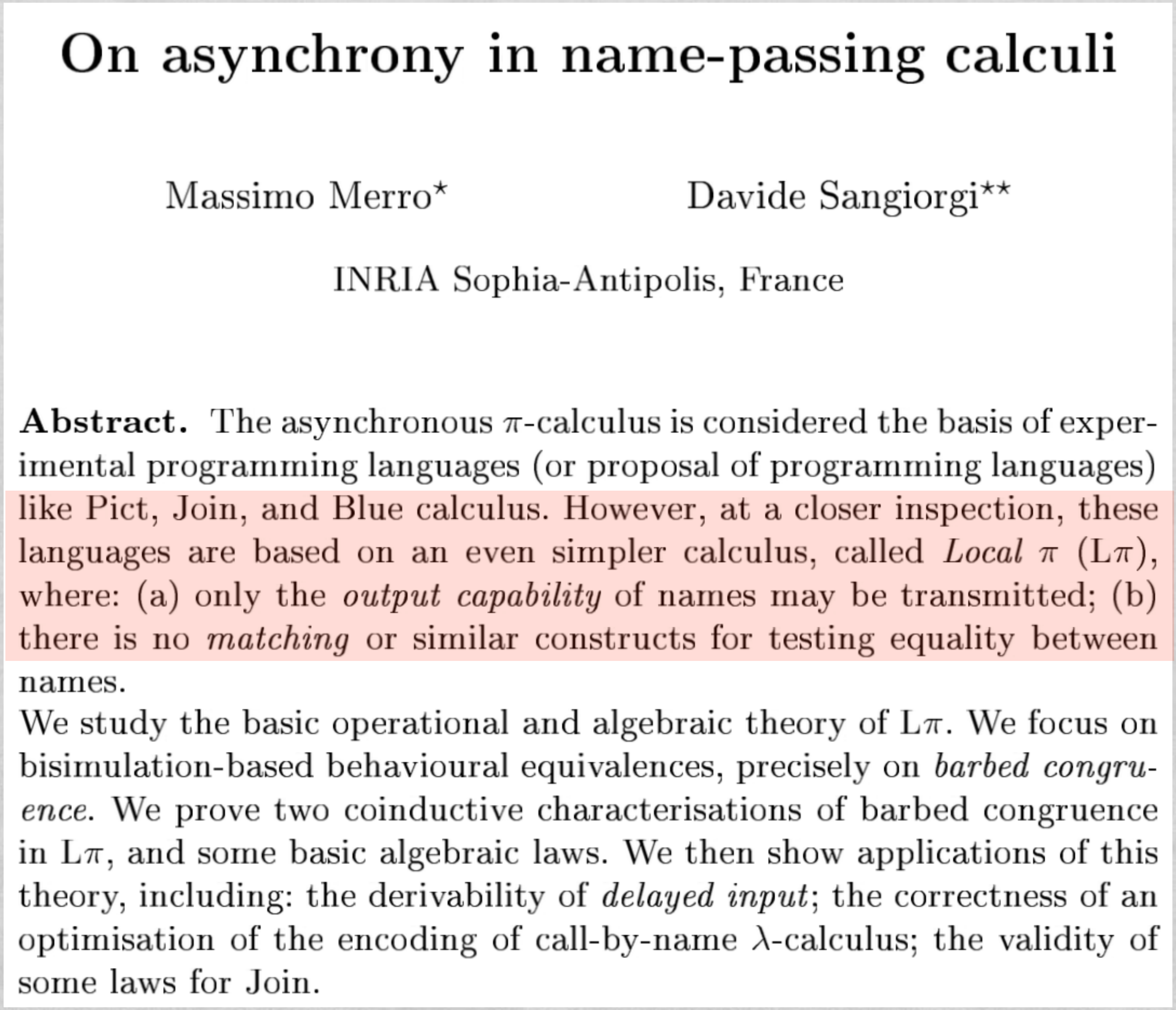
## uniform receptiveness [Sangiorgi97]

- The continuation behaviour for each shared name is **uniform**
- Corresponds to the unique definition of shared servers
- Uniform receptivness [Sangiorgi97] relies on **locality**:

  Only the output capability of shared names is passed around

  Processes forbidden to receive on shared received names
- Allows "efficient" distributed implementations of name passing and routing since no "impersonation" of addresses is possible.
- the **locality property** was studied in [MerroSangiorgi04]

# Locality [MerroSangiorgi04]



## On asynchrony in name-passing calculi

Massimo Merro* Davide Sangiorgi**

INRIA Sophia-Antipolis, France

**Abstract.** The asynchronous $\pi$-calculus is considered the basis of experimental programming languages (or proposal of programming languages) like Pict, Join, and Blue calculus. However, at a closer inspection, these languages are based on an even simpler calculus, called *Local* $\pi$ (L$\pi$), where: (a) only the *output capability* of names may be transmitted; (b) there is no *matching* or similar constructs for testing equality between names.
We study the basic operational and algebraic theory of L$\pi$. We focus on bisimulation-based behavioural equivalences, precisely on *barbed congruence*. We prove two coinductive characterisations of barbed congruence in L$\pi$, and some basic algebraic laws. We then show applications of this theory, including: the derivability of *delayed input*; the correctness of an optimisation of the encoding of call-by-name $\lambda$-calculus; the validity of some laws for Join.

## Duality for All Session Types

$S ::= 1 \mid U \otimes S \mid U \parr S \mid S \oplus S \mid S \& S \mid\ !S \mid\ ?S$

$$\overline{U \otimes S} = U \multimap \overline{S} = \overline{U} \parr \overline{S}$$
$$\overline{U \parr S} = \overline{U} \otimes \overline{S}$$
$$\overline{S \oplus S} = \overline{S} \& \overline{S} \qquad\qquad \overline{\overline{S}} = S$$
$$\overline{S \& S} = \overline{S} \oplus \overline{S}$$
$$\overline{1} = 1$$
$$\overline{!S} = ?\overline{S}$$
$$\overline{?S} = !\overline{S}$$

## Session Types as CLL Propositions

$$S ::= 1 \mid U \otimes S \mid U \parr S \mid S \oplus S \mid S \& S \mid\ !S \mid ?S$$

$$\overline{U \otimes S} = U \multimap \overline{S} = \overline{U} \parr \overline{S}$$

$$\overline{U \parr S} = \overline{U} \otimes \overline{S}$$

$$\overline{S \oplus S} = \overline{S} \& \overline{S}$$

$$\overline{S \& S} = \overline{S} \oplus \overline{S}$$

$$\overline{1} = 1$$

$$\overline{!S} = ?\overline{S}$$

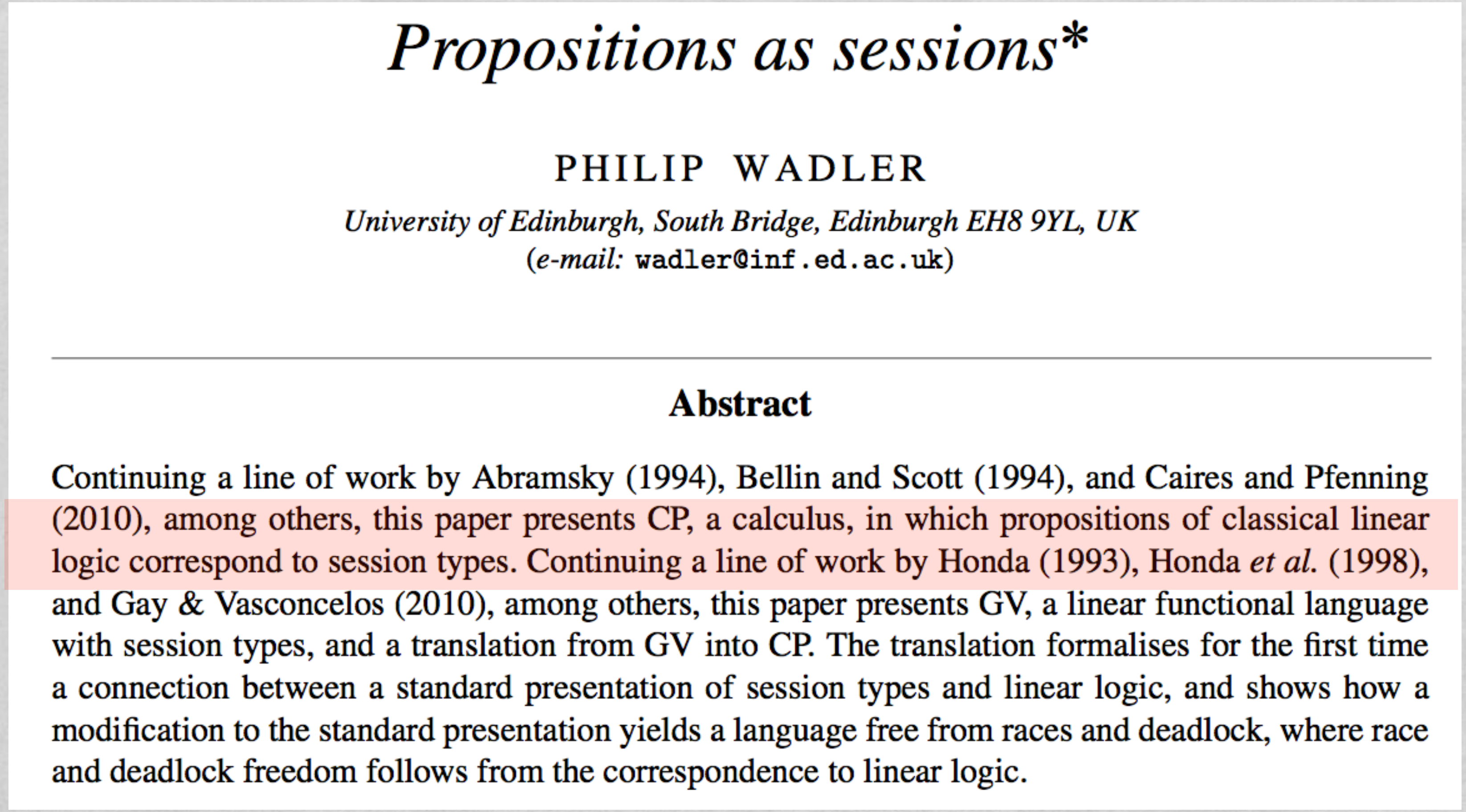
$$\overline{?S} = !\overline{S}$$

$$\overline{\overline{S}} = S$$

$$\overline{S} = S^{\perp}$$

$$U \multimap S = \overline{U} \parr S$$

# Session Types as CLL Propositions



*Propositions as sessions*[*]

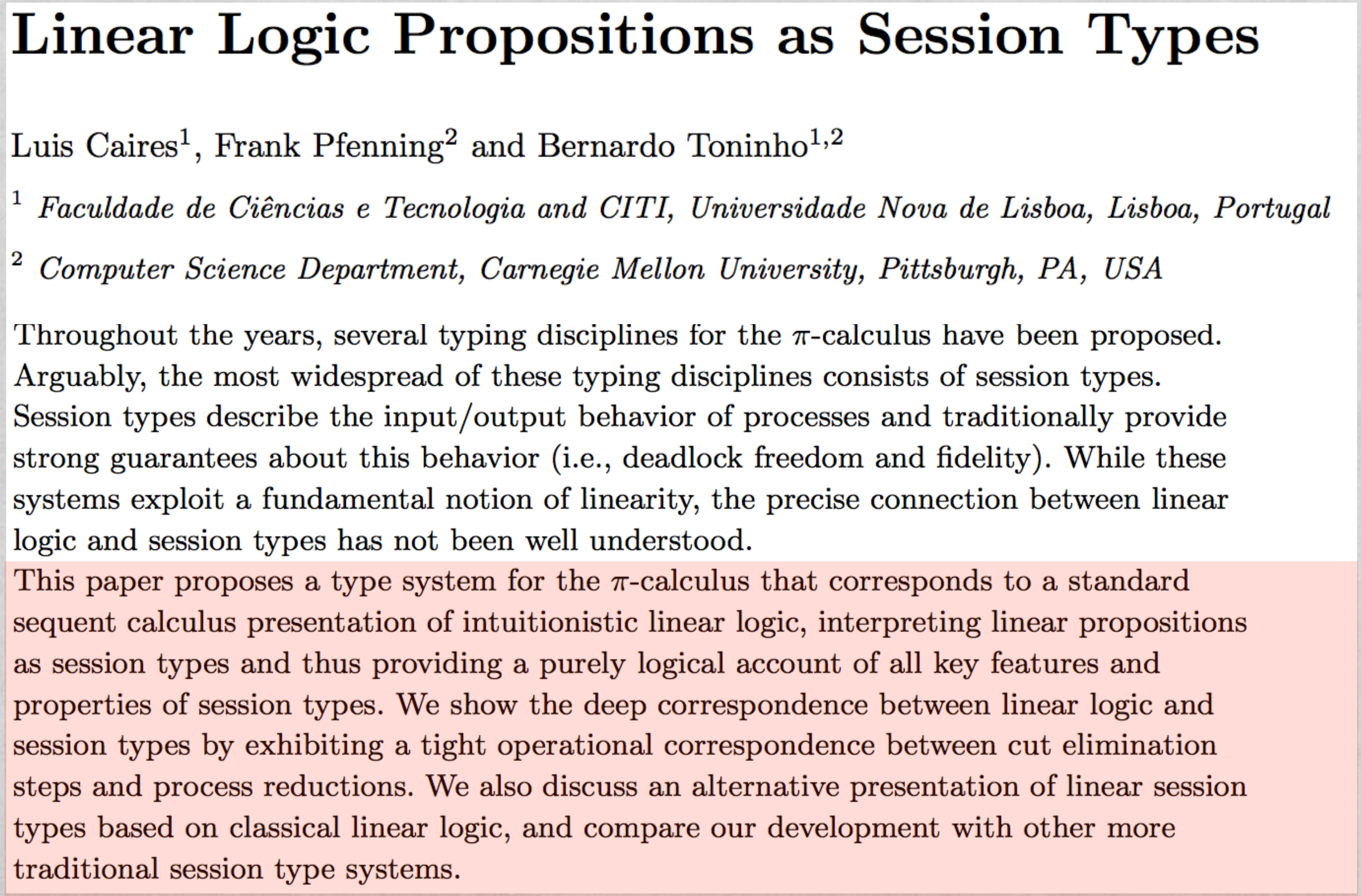
PHILIP WADLER

*University of Edinburgh, South Bridge, Edinburgh EH8 9YL, UK*
(*e-mail:* `wadler@inf.ed.ac.uk`)

**Abstract**

Continuing a line of work by Abramsky (1994), Bellin and Scott (1994), and Caires and Pfenning (2010), among others, this paper presents CP, a calculus, in which propositions of classical linear logic correspond to session types. Continuing a line of work by Honda (1993), Honda *et al.* (1998), and Gay & Vasconcelos (2010), among others, this paper presents GV, a linear functional language with session types, and a translation from GV into CP. The translation formalises for the first time a connection between a standard presentation of session types and linear logic, and shows how a modification to the standard presentation yields a language free from races and deadlock, where race and deadlock freedom follows from the correspondence to linear logic.

# Session Types as CLL Propositions



**Linear Logic Propositions as Session Types**

Luis Caires[1], Frank Pfenning[2] and Bernardo Toninho[1,2]

[1] *Faculdade de Ciências e Tecnologia and CITI, Universidade Nova de Lisboa, Lisboa, Portugal*

[2] *Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA*

Throughout the years, several typing disciplines for the $\pi$-calculus have been proposed. Arguably, the most widespread of these typing disciplines consists of session types. Session types describe the input/output behavior of processes and traditionally provide strong guarantees about this behavior (i.e., deadlock freedom and fidelity). While these systems exploit a fundamental notion of linearity, the precise connection between linear logic and session types has not been well understood.

This paper proposes a type system for the $\pi$-calculus that corresponds to a standard sequent calculus presentation of intuitionistic linear logic, interpreting linear propositions as session types and thus providing a purely logical account of all key features and properties of session types. We show the deep correspondence between linear logic and session types by exhibiting a tight operational correspondence between cut elimination steps and process reductions. We also discuss an alternative presentation of linear session types based on classical linear logic, and compare our development with other more traditional session type systems.

## Classical Linear Logic [Andreolli'90]

$\vdash \Delta; \Theta$

- $\Delta$ linear context (multiset)
- $\Theta$ cartesian context (set)

$$\vdash \bar{A}, A; \Theta \qquad \frac{\vdash \bar{A}, \Delta_1; \Theta \quad \vdash A, \Delta_2; \Theta}{\vdash \Delta_1, \Delta_2; \Theta} \qquad \frac{\vdash \Delta_1; \Theta \quad \vdash \Delta_2; \Theta}{\vdash \Delta_1, \Delta_2; \Theta}$$

$$\vdash 1; \Theta \qquad \frac{\vdash \Delta ; \Theta}{\vdash \Delta, \bot ; \Theta}$$

$$\frac{\vdash \Delta_1, A; \Theta \quad \vdash \Delta_2, B; \Theta}{\vdash \Delta_1, \Delta_2\, A \otimes B; \Theta} \qquad \frac{\vdash \Delta, A, B; \Theta}{\vdash \Delta, A \parr B; \Theta}$$

NB. This system corresponds to a classical version of DILL

## Classical Session Types [CPT'12-14,C14]

$[x \leftrightarrow y] \vdash x{:}\overline{A}, y{:}A; \Theta$

$\mathbf{0} \vdash \ ; \Theta$

$$\frac{Q \vdash x{:}\overline{A}, \Delta_1; \Theta \quad P \vdash x{:}A, \Delta_2; \Theta}{(\mathrm{new}\ x)(Q \mid P) \vdash \Delta_1, \Delta_2; \Theta}$$

$$\frac{Q \vdash \Delta_1; \Theta \quad P \vdash \Delta_2; \Theta}{Q \mid P \vdash \Delta_1, \Delta_2; \Theta}$$

$\overline{\mathtt{close}} \vdash 1; \Theta$

$$\frac{P \vdash \Delta ; \Theta}{\mathtt{close};P \vdash \Delta, \bot ; \Theta}$$

$$\frac{Q \vdash \Delta_1, y{:}A; \Theta \quad P \vdash \Delta_2, x{:}B; \Theta}{\overline{x}(y).(Q \mid P) \vdash \Delta_1, \Delta_2, x{:}A \otimes B; \Theta}$$

$$\frac{P \vdash \Delta, y{:}A, x{:}B; \Theta}{x(y).P \vdash \Delta, x{:}A \parr B; \Theta}$$

## Classical Linear Logic [TCP'12-14]

$$\frac{P \vdash \Delta, x{:}\mathrm{A}; \Theta}{x.\mathtt{inl};P \vdash \Delta, x{:}\mathrm{A}\oplus\mathrm{B}; \Theta}$$

$$\frac{P \vdash \Delta, x{:}\mathrm{B}; \Theta}{x.\mathtt{inr};P \vdash \Delta, x{:}\mathrm{A}\oplus\mathrm{B}; \Theta}$$

$$\frac{Q \vdash \Delta, x{:}\mathrm{B}; \Theta \quad P \vdash \Delta, x{:}\mathrm{B}; \Theta}{x.\mathsf{case}(Q,P) \vdash \Delta, x{:}\mathrm{A}\&\mathrm{B}; \Theta}$$

## Classical Linear Logic [TCP'12-14]

$$\frac{P \vdash y{:}A \,;\, \Theta}{!x(y).P \vdash x{:}!A \,;\, \Theta}$$

$$\frac{P \vdash \Delta \,;\, x{:}A, \Theta}{P \vdash \Delta, x{:}?A;\, \Theta}$$

$$\frac{P \vdash \Delta, y{:}A \,;\, x{:}A, \Theta}{\overline{x}(y).P \vdash \Delta \,;\, x{:}A, \Theta}$$

$$\frac{Q \vdash y{:}\overline{A} \,;\, \Theta \qquad P \vdash \Delta \,;\, x{:}A, \Theta}{(\text{new } x)(!x(y).Q \mid P) \vdash \Delta \,;\, \Theta}$$

# Replication Reduction

$$\frac{P \vdash y{:}\bar{A} \,;\, \Theta \qquad \dfrac{Q \vdash \Delta, y{:}A \,;\, x{:}A, \Theta}{\bar{x}(y).Q \vdash \Delta \,;\, x{:}A, \Theta}}{(\mathsf{new}\ x)(!x(y).P \mid \bar{x}(z).Q) \vdash \Delta \,;\, \Theta}$$

$$\rightarrow$$

$$\frac{P \vdash y{:}\bar{A} \,;\, \Theta \qquad \dfrac{P \vdash y{:}\bar{A} \,;\, \Theta \qquad Q \vdash \Delta, y{:}A \,;\, x{:}A, \Theta}{\Gamma; \Delta, y{:}A \vdash (\mathsf{new}\ x)(!x(y).P \mid Q) :: C}}{\Gamma; \Delta \vdash (\mathsf{new}\ y)(P \mid (\mathsf{new}\ x)(!x(y).P \mid Q)) :: C}$$

# Proofs = Processes

| | | | |
|---|---|---|---|
| $P$ | $::=$ | $0$ | (inaction) |
| | $\mid$ | $[x \leftrightarrow y]$ | (forwarder) |
| | $\mid$ | $(\mathtt{new}\ x)(P \mid Q)$ | (composition) |
| | $\mid$ | $x(y).P$ | (input) |
| | $\mid$ | $\bar{x}(y).P$ | (output) |
| | $\mid$ | $!x(y).P$ | (shared server) |
| | $\mid$ | $x.\mathtt{case}(P,Q)$ | (offer) |
| | $\mid$ | $x.\mathtt{inl};Q$ | (choose left) |
| | $\mid$ | $x.\mathtt{inr};Q$ | (choose right) |
| | $\mid$ | $x.\mathtt{close};Q$ | (wait) |
| | $\mid$ | $x.\overline{\mathtt{close}}$ | (close) |

## Proof Conversions = Process Identities

- Structural Conversions ( $\equiv$ )

  ( $\equiv$ ) matched by π structural congruence ( $\equiv$ )
- Computational Conversions ( $\rightarrow$ )

  ( $\rightarrow$ ) matched by π reduction ( $\rightarrow$ )
- Structural Conversions ( $\simeq$ )

  ( $\simeq$ ) matched by typed π observational equivalence ( $\equiv$ )
- All Conversions ( $\cong$ )

# Proof Conversions = Process Identities

Structural Conversions ( $\equiv$ )

Identify structurally identical proofs (e.g, commute cuts, expose redexes)

Correspond to standard structural congruences ( $\equiv$ )

$0 \mid P \;\equiv\; P$ cut/mix conversions

$(\text{new } x)(P \mid (\text{new } y)(Q \mid R)) \equiv (\text{new } y)((\text{new } x)(P \mid Q) \mid R)$

$(\text{new } x)(P \mid (\text{new } y)(Q \mid R)) \equiv (\text{new } y)(Q \mid (\text{new } x)(P \mid R))$

$(\text{new } x)(P \mid (Q \mid R)) \equiv Q \mid (\text{new } x)(P \mid R)$ cut/mix conversions

# Proof Reductions = Process Reductions

Computational Conversions ($\rightarrow$)

Reduce proofs into simpler ones (e.g, decreases types)

correspond to standard process reductions ( $\rightarrow$ )

$(\mathsf{new}\ x)(x.\overline{\mathtt{close}} \mid x.\mathtt{close}.P) \rightarrow P$

$(\mathsf{new}\ x)(\bar{x}(y).(P|Q) \mid x(y).R) \rightarrow (\mathsf{new}\ y)(P \mid (\mathsf{new}\ x)(Q \mid R))$

$(\mathsf{new}\ x)(x.\mathtt{case}(Q,P) \mid x.\mathtt{inl};R) \rightarrow (\mathsf{new}\ x)(Q \mid R)$

$(\mathsf{new}\ x)(!x(y).P \mid \bar{x}(y).Q) \rightarrow (\mathsf{new}\ y)(P \mid (\mathsf{new}\ x)(!x(y).P \mid Q))$

## Proof Conversions = Process Identities

- Structural Conversions (≃)

  Correspond to well known typed strong bisimilarities ( $\simeq$ )

$(\text{new } x)(!x(y).P \mid (\text{new } z)(Q \mid R)) \simeq$
$\quad (\text{new } z)(\ (\text{new } x)(!x(y).P \mid Q) \mid (\text{new } x)(!x(y).P \mid R))$

$(\text{new } x)(!x(y).P \mid (\text{new } z)(!z(u).Q \mid R)) \simeq$
$\quad (\text{new } z)(\ !z(u).(\text{new } x)(!x(y).P \mid Q) \mid (\text{new } x)(!x(y).P \mid R))$

$(\text{new } x)(!x(y).P \mid Q) \simeq Q\ \ [\ x \notin fn(Q)\ ]$

- The **sharpened replication lemmas** of [SangiorgiWalker01].

## Proof Conversions = Process Identities

- Structural Conversions (≃)

  Correspond to well known typed strong bisimilarities ( $\simeq$ )

$(\text{new } x)(!x(y).P \mid Q \mid R) \simeq$
$\quad (\text{new } x)(!x(y).P \mid Q) \mid (\text{new } x)(!x(y).P \mid R)$

cut/mix conversions

$(\text{new } x)(!x(y).P \mid (\text{new } z)(!z(u).Q \mid R)) \simeq$
$\quad (\text{new } z)(\ !z(u).(\text{new } x)(!x(y).P \mid Q) \mid (\text{new } x)(!x(y).P \mid R))$

$(\text{new } x)(!x(y).P \mid Q) \simeq Q \ \ [\ x \notin fn(Q)\ ]$

- The **sharpened replication lemmas** of [SangiorgiWalker01].

# CLL is non-local

$SendBroad(a) \triangleq \overline{a}(q).\ (\overline{q}(v_1).\overline{q}(v_2).\mathbf{0} \mid Q)$

$System \triangleq (\mathsf{new}\ a)(\ SendBroad(a) \mid a(x).!x(s).P))$

$a(x).!x(s).P \vdash a\colon \mathbf{?}\mathrm{A}\multimap\mathbf{?}\mathrm{B}\ ;\ \text{-}$

$\overline{q}(v_1).\overline{q}(v_2).\mathbf{0} \mid \overline{q}(v_3).\mathbf{0} \vdash q\colon \mathbf{?}\mathrm{A}\ ;\ \text{-}$ $\qquad Q \vdash a{:}!\overline{\mathrm{B}}$

$SendBroad(a) \vdash a\colon \mathbf{?}\mathrm{A} \otimes !\overline{\mathrm{B}}\ ;\ \text{-}$ $\qquad System \vdash \text{-}\ ;\ \text{-}$

- Unlike DILL, CLL allows us to express full duality on shared sessions, by dropping the (too strict) locality property.
- Remarkably, the classical type structure still ensures uniform receptiveness on shared names (thus confluence, no surprise)

# CLL ensures uniform ω-receptiveness

$SendBroadW(a) \triangleq \overline{a}(q).\ (\overline{q}(v).p[q].Q \mid P)$

$\vdash SendBroadW(a) :: a{:}?\mathrm{A}\otimes\mathrm{B},\ p{:}!\overline{\mathrm{A}}\otimes 1;\ \Theta$

$\vdash \overline{q}(v).p[q].Q :: q{:}\ ?\mathrm{A},\ p{:}!\overline{\mathrm{A}}\otimes 1;\ \Theta$

$\vdash \overline{p}[q].Q :: p{:}!\mathrm{A}\otimes\overline{1};\ q{:}\mathrm{A},\Theta$

$\vdash \overline{p}(h).!h(z).\overline{q}(k).[k\leftrightarrow z] :: p{:}!\overline{\mathrm{A}}\ ;\ q{:}\mathrm{A},\Theta$

- Typing allows the receptive endpoint $q^-$ to be sent (on $a$) at type ?A, linearly (exactly once), leading to a “unique server”.
- Typing enforces all positive uses of $q$ ($q^+$) to be sent only at type $!\overline{\mathrm{A}}$, mediated by a proxy (via !R)

## Building up

- Behavioural Polymorphism and Parametricity
- Dependent types
- Asynchrony
- Authorisation
- LNL and Higher-Order processes
- Logical Relations
- Encoding Multiparty Systems
- Non-determinism (forthcoming)

## Behavioral Polymorphism

## Behavioral Polymorphism

- Polymorphism (aka "generics") is an indispensable feature in everyday programming, say Java

  ```
  class LinkedList<T>
  ```

  T is a type parameter than can be instantiated (at compile time) by a given type (say, class or interface)
- Parametric polymorphism was introduced in PL by Reynolds and is linked by the Curry-Howard correspondence to quantification in second-order logic by Girard
- Repeating the exercise on logical session types we discover a powerful notion of **behavioural polymorphism**, just too hard to tackle by extant techniques [Turner,PierceSangiorgi]

## simply typed λ-calculus [Church30]

$$\Gamma, x{:}\mathrm{A} \vdash x{:}\mathrm{A}$$

$$\frac{\Gamma, x : \mathrm{A} \vdash M : \mathrm{B}}{\Gamma \vdash \lambda x{:}\mathrm{A}.M : \mathrm{A}\to\mathrm{B}} \qquad \frac{\Gamma \vdash M : \mathrm{A}\to\mathrm{B} \quad \Gamma \vdash N : \mathrm{A}}{\Gamma \vdash MN : \mathrm{B}}$$

Tapp(Tlam([$x$]$d_1$),$d_2$) $\to d_1\{d_2/x\}$

## Polymorphic λ-calculus [Girard-Reynolds]

$\Omega \vdash M$ **ty**

$\Omega; \Gamma \vdash M{:}A$

$$\frac{\Omega, X; \Gamma \vdash M : B}{\Omega; \Gamma \vdash \lambda X.M : \forall X.B}$$

$$\frac{\Omega; \Gamma \vdash M : \forall X.B \quad \Omega \vdash S \ \mathbf{ty}}{\Omega; \Gamma \vdash MS : B\{S/X\}}$$

$\mathtt{TTapp}(\mathtt{TTlam}([X]d_1), S) \rightarrow d_1\{X/S\}$

## Polymorphic λ-calculus [Girard-Reynolds]

$$\frac{\Omega;\Gamma \vdash M : B\{S/X\} \quad \Omega \vdash S\ \mathbf{ty}}{\Omega;\Gamma \vdash \langle S, M\rangle : \exists X.B} \qquad \frac{\Omega;\Gamma \vdash M : \exists X.B \quad \Omega,X;,\ x{:}X \vdash N{:}A}{\Omega;\Gamma \vdash \mathtt{let}\ \langle X,x\rangle = M\ \mathtt{in}\ N{:}A}$$

$$\mathtt{TTopen}(\mathtt{TThide}[X](d_1,S),d_2) \to d_2\{X/S,x/d_1\}$$

# Linear Propositions as Session Types

- Typing judgement

$$\Omega;\Gamma;\Delta \vdash P :: y{:}\mathrm{C}$$

- Intuition: judgement states a rely-guarantee property:

for all session types Ω, whenever composed with processes offering a session $\mathrm{A}_i$ at $x_n$, *P* offers a session of type C at $y$

$$\frac{\Omega;\Gamma;\Delta_1 \vdash Q :: x{:}\mathrm{A} \quad \Omega;\Gamma;\Delta_2, x{:}\mathrm{A} \vdash P :: \mathrm{C}}{\Omega;\Gamma;\Delta_1, \Delta_2 \vdash (\mathsf{new}\ x)(Q \mid P) :: \mathrm{C}}$$

typing ensures fidelity and global progress (cut-elimination)

# Proofs = Processes

| | | | |
|---|---|---|---|
| $P$ | $::=$ | $0$ | (inaction) |
| | $\mid$ | $[x \leftrightarrow y]$ | (linear forwarder) |
| | $\mid$ | $(\mathtt{new}\ x)(P \mid Q)$ | (composition) |
| | $\mid$ | $x(y).P$ | (input) |
| | $\mid$ | $\bar{x}(y).P$ | (output) |
| | $\mid$ | $!x(y).P$ | (shared server) |
| | $\mid$ | $x.\mathtt{case}(P,Q)$ | (offer) |
| | $\mid$ | $x.\mathtt{inl};Q$ | (choose left) |
| | $\mid$ | $x.\mathtt{inr};Q$ | (choose right) |
| | $\mid$ | $x[\mathrm{S}].P$ | (type output) |
| | $\mid$ | $x(\mathrm{X}).Q$ | (type input) |

# Linear Propositions as Session Types

$$\frac{\Omega \vdash S\ \mathbf{ty} \quad \Omega;\Gamma;\Delta \vdash P :: x{:}B\{S/X\}}{\Omega;\Gamma;\Delta \vdash x[S].P :: x{:}\exists X.B} \qquad \frac{\Omega, X;\Gamma;\Delta, x{:}B \vdash P :: C}{\Omega;\Gamma;\Delta, x{:}\exists X.B \vdash x(X).P :: C}$$

$$\frac{\Omega \vdash S\ \mathbf{ty} \quad \Gamma;\Delta, x{:}B\{S/X\} \vdash P :: C}{\Gamma;\Delta, x{:}\forall X.B \vdash x[S].P :: C} \qquad \frac{\Omega, X;\Gamma;\Delta \vdash P :: x{:}B}{\Omega;\Gamma;\Delta \vdash x(X).P :: x{:}\forall X.B}$$

# Type Send and Receive

$$\dfrac{\dfrac{\Omega, X; \Gamma; \Delta_1 \vdash P :: x{:}B}{\Omega; \Gamma; \Delta_1 \vdash x(X).P :: x{:}\forall X.B} \qquad \dfrac{\Omega \vdash S\ \mathbf{ty} \quad \Gamma; \Delta_2, x{:}B\{S/X\} \vdash Q :: C}{\Omega; \Gamma; \Delta_2, x{:}\forall X.B \vdash x[S].Q :: C}}{\Omega; \Gamma; \Delta_1, \Delta_2 \vdash (\mathsf{new}\ x)(x(X).P \mid x[S].Q) :: C}$$

$$\rightarrow$$

$$\dfrac{\Omega; \Gamma; \Delta_1 \vdash P\{S/X\} :: x{:}B\{S/X\} \qquad \Omega; \Gamma; \Delta_2, x{:}B\{S/X\} \vdash Q :: C}{\Omega; \Gamma; \Delta_1, \Delta_2 \vdash (\mathsf{new}\ x)(P\{S/X\} \mid Q) :: C}$$

$\mathsf{Tcut}[x](\mathsf{TR}\forall[X](d_1), \mathsf{TL}\forall(S, d_2)) \rightarrow \mathsf{Tcut}[x](d_1\{S/X\}, d_2)$

# Type Send and Receive

$$\dfrac{\dfrac{\Omega \vdash S\ \mathbf{ty} \quad \Omega;\Gamma;\Delta_1 \vdash P :: x{:}B\{S/X\}}{\Omega;\Gamma;\Delta_1 \vdash x[S].P :: x{:}\exists X.B} \qquad \dfrac{\Omega,X;\Gamma;\Delta_2, x{:}B \vdash Q :: C}{\Omega;\Gamma;\Delta_2, x{:}\exists X.B \vdash x(X).Q :: C}}{\Omega;\Gamma;\Delta_1,\Delta_2 \vdash (\mathsf{new}\ x)(x[S].P \mid x(X).Q) :: C}$$

$$\rightarrow$$

$$\dfrac{\Omega;\Gamma;\Delta_1 \vdash P :: x{:}B\{S/X\} \qquad \Omega;\Gamma;\Delta_2, x{:}B\{S/X\} \vdash Q\{S/X\} :: C}{\Omega;\Gamma;\Delta_1,\Delta_2 \vdash (\mathsf{new}\ x)(P \mid Q\{S/X\}) :: C}$$

$\mathsf{Tcut}[x](\mathsf{TR}\exists(S, d_1), \mathsf{TL}\exists[X](d_2)) \rightarrow \mathsf{Tcut}[x](d_1, d_2\{S/X\})$

# Classical Typing Rules

$$\frac{P \vdash \Delta, x{:}A;\ \Theta;\ \Omega, X}{x(X).P \vdash \Delta, x{:}\forall X.A;\ \Theta;\ \Omega}$$

$$\frac{\Omega \vdash S\ \mathbf{ty}\quad P \vdash \Delta, x{:}B\{S/X\};\ \Theta;\ \Omega}{x[S].P \vdash \Delta, x{:}\exists X.B;\ \Theta;\ \Omega}$$

## A Cloud Computing Server

# The Generic Cloud Service

$\mathrm{API} \triangleq\ !\&\{\ \mathtt{rmov}:(\mathit{Name} \multimap \mathrm{MP4} \otimes 1),\ \mathtt{wmov}:(\mathit{Name} \multimap \mathrm{MP4} \multimap 1))$

$\mathit{CloudServer} \triangleq \forall X.!(\mathrm{API} \multimap X) \multimap\ !X$

$\mathit{CS}(a) \triangleq a(Y).a(t).!a(w).\overline{t}(s).\overline{s}(ap).([ap \leftrightarrow api] \mid [s \leftrightarrow w])$

- $;\ api{:}\mathrm{API} \vdash \mathit{CS}(a) :: a{:}\mathit{CloudServer}$

- $;\ - \vdash \mathrm{MDB}(api) :: api{:}\mathrm{API}$

- $;\ - \vdash (\mathtt{new}\ api)(\mathrm{MDB}(api) \mid \mathit{CS}(a)) :: a{:}\mathit{CloudServer}$

# Uploading Service to the Cloud

$\mathrm{API} \triangleq\ !\&\{\ \mathtt{rmov}{:}(\mathit{Name} \multimap \mathrm{MP4}\otimes 1),\ \mathtt{wmov}{:}(\mathit{Name} \multimap \mathrm{MP4} \multimap 1)\}$

$\mathit{MCode}(s,\mathit{api}) \triangleq s(\mathit{title}).\overline{\mathit{api}}(h).h.\mathtt{rmov};\overline{h}(\mathit{title}).h(\mathit{mfile}).\overline{s}(\mathit{mfile}).\mathbf{0}$

$\mathit{UserProto} \triangleq \mathit{Name} \multimap \mathrm{MP4}\otimes 1$

$-\ ;\ - \vdash s(\mathit{api}).\mathit{MCode}(s) :: s{:}\ \mathrm{API} \multimap \mathit{UserProto}$

$\mathit{ServiceCode}(t) \triangleq\ !t(s).s(\mathit{api}).\mathit{MCode}(s,\mathit{api})$

$-\ ;\ - \vdash \mathit{ServiceCode}(t) :: t{:}\ !(\mathrm{API} \multimap \mathit{UserProto})$

# Creating a Custom Service

$- ; - \vdash (\textsf{new}\ api)(\mathrm{MDB}(api) \mid CS(a)) :: a{:}CloudServer$

$FreeViewProto(n) \triangleq a[UserProto].\overline{a}(t).(ServiceCode(t) \mid [a{\leftrightarrow}n])$

$- ; a{:}CloudServer \vdash FreeViewProto(n) :: n{:}!UserProto$

$FreeOnCloud \triangleq (\textsf{new}\ a)(CloudServer \mid FreeViewProto(n))$

$- ; - \vdash FreeOnCloud :: n{:}!UserProto$

# Creating a Custom Service

$- ; - \vdash (\text{new } api)(\mathrm{MDB}(api) \mid CS(a)) :: a{:}CloudServer$

$FreeViewProto(n) \triangleq a[UserProto].\overline{a}(t).(ServiceCode(t) \mid [a \leftrightarrow n])$

$- ; a{:}CloudServer \vdash FreeViewProto(n) :: n{:}!UserProto$

$FreeOnCloud \triangleq (\text{new } a)(CloudServer \mid FreeViewProto(n))$

$- ; - \vdash FreeOnCloud :: n{:}!UserProto$

$Isabel(n) \triangleq \overline{n}(a).\overline{a}(\text{“}\mathbf{interstellar}\text{”}).a(file).\dots$

$- ; n{:}!UserProto \vdash Isabel(n) :: p{:}Fun$

$- ; - \vdash (\text{new } n)(FreeOnCloud \mid Isabel(n))) :: p{:}Fun$

## Logical Relations and Parametricity

- Being based on logic, our systems are amenable to well-known reasoning techniques that can be used to establish important meta properties.
- We have developed (linear) logical relations and associated proof techniques for our session type systems [ESOP12, ESOP13, TGC14, BT15], addressing strong normalisation, observational equivalences, parametricity.
- N.B: Logical relations have been originally introduced by [Tait58], but are currently a basic tool for studying general semantic properties enforced by type systems [see A13].

# A Logical Predicate $T_\eta{}^\omega[\![z{:}A]\!]$

$P \in T_\eta{}^\omega[\![z{:}X]\!] \triangleq P \in \eta(X)(z)$

$P \in T_\eta{}^\omega[\![z{:}1]\!] \triangleq \forall Q.\ (P \Rightarrow Q \wedge Q \rightarrow\ ) \supset Q \equiv_! \mathbf{0}$

$P \in T_\eta{}^\omega[\![z{:}A \multimap B]\!] \triangleq \forall Q.(P \overset{z(y)}{\Rightarrow} Q) \supset$

$\quad \forall R \in T_\eta{}^\omega[\![y{:}A]\!].\ (\text{new } y)(R \mid Q) \in T_\eta{}^\omega\, [\![z{:}B]\!]$

$P \in T_\eta{}^\omega[\![z{:}A \otimes B]\!] \triangleq \forall Q.(P \overset{\overline{z}(y)}{\Rightarrow} Q) \supset$

$\quad \exists P_1, P_2.\ P \equiv_! (P_1 \mid P_2) \wedge P_1 \in T_\eta{}^\omega\, [\![y{:}A]\!] \wedge P_2 \in T_\eta{}^\omega\, [\![z{:}B]\!]$

$P \in T_\eta{}^\omega[\![z{:}\forall X.A]\!] \triangleq \forall S, P', R[{:}S].\ (P \overset{z(S)}{\Rightarrow} Q) \supset Q \in T_{\eta[X/R[:S]]}{}^{\omega[X/S]}[\![z{:}A]\!]$

$P \in T_\eta{}^\omega[\![z{:}\exists X.A]\!] \triangleq \exists S, P', R[{:}S].\ (P \overset{z[S]}{\Rightarrow} Q) \supset Q \in T_{\eta[X/R[:S]]}{}^{\omega[X/S]}[\![z{:}A]\!]$

# Logical Candidate

- A logical candidate $R[\![z{:}A]\!]$ is a set of processes such that:

  $P \in R[\![z{:}A]\!]$ *implies* $-;-;- \vdash P :: z{:}A$

  $P \in R[\![z{:}A]\!]$ *implies* $P$ *strongly terminates under* $\rightarrow$

  $P \in R[\![z{:}A]\!]$ *and* $P \equiv_! Q$ *implies* $Q \in R[\![z{:}A]\!]$

  $P \in R[\![z{:}A]\!]$ *and* $P \Rightarrow Q$ *implies* $Q \in R[\![z{:}A]\!]$

  $P \in R[\![z{:}A]\!]$ if *for all* $Q$ *such that* $P \Rightarrow Q$ *we have* $Q \in R[\![z{:}A]\!]$

- The defined notion of candidate [Girard] captures the intended semantic property here, in this case termination.

# Strong Termination

**Theorem**.

For all $\omega{:}\Omega$ $\eta{:}\Omega$, $T_\eta^\omega[\![z{:}A]\!]$ is a logical candidate $R[\![z{:}\ \omega(A)]\!]$

**Theorem**.

If $\Omega;\Gamma;\Delta \vdash P{::}\ y{:}C$ and $\omega{:}\Omega$, $\eta{:}\Omega$ then $\omega(P) \in T_\eta^\omega[\![\ \Omega;\Gamma;\Delta\vdash\ P{::}\ y{:}C]\!]$

**Theorem**.

If $\Omega;\Gamma;\Delta \vdash P{::}\ y{:}C$ and $\omega{:}\Omega$ then $\omega(P)$ strongly terminates under $\rightarrow$

# Logical Relations and Parametricity

- Parametricity states that polymorphic code operates in a completely uniform way across all type instantiations
- Traditionally, parametricity is important to establish e.g., representation independence or security properties of ADTs.
- In [PCPT'13-ESOP] we have developed a powerful theory of parametricity for polymorphic session types.
- We show e.g., how observational equivalence of two restaurant finding apps relying on completely different map services (with very different interaction protocols).
- Simple type based analysis technique shows that no client can tell which map service is being used "under the hood".

## Interface Contracts and Assertions

## Interface Contracts and Assertions

- Session types just talk about the abstract communication behaviour, but richer behavioural specifications will definitely need to talk about properties of exchanged data as well
- Traditionally, this involves considering notions of “contracts” or “assertions”, in the spirit of axiomatic semantics [Hoare].
- Along this lines, [BHTY10] studied one possible combination of multiparty session types with FOL pre / post conditions.

## Interface Contracts and Assertions

- Following the Curry-Howard approach we may naturally integrate session types (propositional linear logic) towards a dependent type theory (intuitionistic first-order logic).
- N.B. while basic values can be encoded as processes, we have no perspective on how to define a consolidated type theory for processes both as behaviours and as values that would support a proper dependent type theory.
- We now illustrate a typed integration of processes, intuitionistic data types, proofs, and “processes as data” inspired by the Mixed Linear-Non-Linear logic of Benton.

# Mixed linear-non-linear Logic [Benton]

$\Psi; \Gamma \vdash M : A$

$\Psi; \Gamma; \Delta \vdash P :: z{:}S$

- $\Delta$ linear channel context (multiset)
- $\Gamma$ cartesian channel context (set)
- $\Psi$ cartesian value context (set)

$$
\begin{aligned}
A ::= {} & \mathbf{int} \mid \mathbf{bool} \mid \mathbf{nat} \mid \text{string} \mid \ldots \\
\mid {} & A \rightarrow B \\
\mid {} & A \wedge B \\
\mid {} & A \vee B \\
\mid {} & \{\, z{:}S \,\} \\
\mid {} & \forall x{:}A.B \\
\mid {} & \exists x{:}A.B
\end{aligned}
$$

$$
\begin{aligned}
S ::= {} & U \otimes S \mid U \multimap S \\
\mid {} & S \oplus S \mid S \& S \\
\mid {} & !U \mid 1 \\
\mid {} & \$A \\
\mid {} & \forall x{:}A.S \\
\mid {} & \exists x{:}A.B
\end{aligned}
$$

# Mixed linear-non-linear Logic [Benton]

$\Psi; \Gamma \vdash M : A$

$\Psi ; \Gamma; \Delta \vdash P :: z{:}S$

- $\Delta$ linear channel context (multiset)
- $\Gamma$ cartesian channel context (set)
- $\Psi$ cartesian value context (set)

$$\frac{\Psi ; \Gamma \vdash M : A}{\Psi ; \Gamma; - \vdash [z \leftarrow M] :: z{:}\$A}$$

$$\frac{\Psi, z{:}A; \Gamma; \Delta \vdash P : C}{\Psi ; \Gamma; \Delta, z{:}\$A \vdash P :: C}$$

$$\frac{\Psi ; \Gamma; - \vdash P :: z{:}S}{\Psi ; \Gamma \vdash \{P\} : \{z{:}S\}}$$

$$\frac{\Psi ; \Gamma \vdash M : \{z{:}S\} \quad \Psi ; \Gamma; \Delta, z{:}S \vdash Q :: C}{\Psi ; \Gamma; \Delta \vdash \mathtt{spawn.}\ z\ (M \mid\mid Q) : C}$$

# Certifying Session Interfaces

"Standard" Session Type (talks about behaviour)

$BankST \triangleq \&\{\, \texttt{with}: \mathbf{nat} \otimes \mathbf{nat} \multimap \&\{\, \texttt{ok};1, \texttt{ko};1\},$

$\qquad\qquad \texttt{deposit}: \mathbf{nat} \otimes \&\{\, \texttt{ok};1,\ \texttt{ko};1\}\ \}$

Dependent Session Type (talks about behaviour + data exchanged)

$BankCI \triangleq \&\{\, \texttt{with}: \exists b{:}\mathbf{nat}.\ \forall v{:}\mathbf{nat}. \forall p{:}\ [v \leq b].\ \&\{\, \texttt{ok};1, \texttt{ko};1\},$

$\qquad\qquad \texttt{deposit}{:} \forall v{:}\mathbf{nat}.\ \forall p{:}\ [0 \leq v].\ \&\{\, \texttt{ok};1, \texttt{ko};1\}\}$

# Dependent Session Types

$$\frac{\Psi;\ \Gamma \vdash M : \mathrm{A} \quad \Psi;\Gamma;\ \Delta \vdash P :: x{:}\mathrm{S}\{M/x\}}{\Psi;\ \Gamma;\ \Delta \vdash x[M].P :: x{:}\ \exists x{:}\mathrm{A.S}}$$

$$\frac{\Psi, y{:}\mathrm{A};\Gamma;\Delta \vdash P :: x{:}\mathrm{S}}{\Psi;\Gamma;\Delta \vdash x(y).P :: x{:}\ \forall y{:}\mathrm{A.S}}$$

# Certifying Session Interfaces

$BankCI \triangleq \&\{$ `with`: $\exists b{:}\mathbf{nat}.\ \forall v{:}\mathbf{nat}.\forall p{:}\ [v \le b].\ \&\{$ ok;1, ko;1 $\}$,

`deposit`: $\forall v{:}\mathbf{nat}.\ \forall p{:}\ [0 < v].\ \&\{$ ok;1, ko;1 $\}\}$

$Client(b) \triangleq b$.`with`.s$(bv)$.s$(bv/2)$.s[**ltehalf**$(bv)$].ok;1

$\Psi; \Gamma; b{:}\ BankCI \vdash Client(b)\ \ ::\ \text{-}\ 1$

$\Psi$ contains a binding for **ltehalf**: $\forall b{:}\mathbf{nat}.\ b/2 \le b$

# Mixed linear-non-linear Logic [Benton]

$\Psi; \Gamma \vdash M : A$

$\Psi ;\Gamma; \Delta \vdash P :: z{:}S$

- $\Delta$ linear channel context (multiset)
- $\Gamma$ cartesian channel context (set)
- $\Psi$ cartesian value context (set)

$$\frac{\Psi ;\Gamma \vdash M : A}{\Psi ;\Gamma; - \vdash [z \leftarrow M] :: z{:}\$A} \qquad \frac{\Psi, z{:}A; \Gamma; \Delta \vdash P : C}{\Psi ;\Gamma; \Delta, z{:}\$A \vdash P :: C}$$

$$\frac{\Psi ;\Gamma; - \vdash P :: z{:}S}{\Psi ;\Gamma \vdash \{P\} : \{z{:}S\}} \qquad \frac{\Psi ;\Gamma \vdash M : \{z{:}S\} \quad \Psi ;\Gamma; \Delta, z{:}S \vdash Q :: C}{\Psi ;\Gamma; \Delta \vdash \texttt{spawn}.\ z\ (M \,||\, Q) : C}$$

# App Store

$$
\begin{aligned}
AppStore \triangleq \&\{\, & \mathtt{game}\colon \{\, g\colon API \multimap Game \},\\
& \mathtt{maps}\colon \{\, g\colon API \multimap GPS \multimap Maps \,\}\\
& \mathtt{cam}\colon \{\, g\colon API \multimap CAM \multimap Cam \} \,\}
\end{aligned}
$$

*Cam* ≜ *… some session type describing the camera App behaviour*

# A toy App Store

$$\mathit{AppStore} \triangleq \&\{\ \texttt{game}\!:\{\ g\!: \mathit{API} \multimap \mathit{Game}\},$$
$$\texttt{maps}\!:\ \{\ g\!: \mathit{API} \multimap \mathit{GPS} \multimap \mathit{Maps}\ \}$$
$$\texttt{cam}\!:\{\ g\!: \mathit{API} \multimap \mathit{CAM} \multimap \mathit{Cam}\}\ \}$$

$\mathit{Cam} \triangleq$ *… some session type describing the camera App behaviour*

$\mathit{Betty}(as,gps) \triangleq$

$$as.\texttt{maps}.as(code).\texttt{spawn}\ g.\ (code \mid \overline{g}(api).\overline{g}(gps).[g{\leftrightarrow}c])\!: c\!:\!\mathit{Maps}$$

$$as\!: \mathit{AppStore},\ api\!:\!\mathit{GPS} \vdash \mathit{Betty}(as,api) :: c\!: \mathit{Maps}$$

## The Cloud Server Type (redux)

$\mathrm{API} \triangleq\ !\&\{\ \mathtt{rmov}{:}(\mathit{Name} \multimap \mathrm{MP4} \otimes 1),$

$\qquad\qquad \mathtt{wmov}{:}(\mathit{Name} \multimap \mathrm{MP4} \multimap 1)\}$

$\mathit{CloudServer} \triangleq \forall \mathrm{X}.\{c{:}\mathrm{API} \multimap \mathrm{X}\} \multimap\ !\mathrm{X}$

## Building up

- Behavioural Polymorphism and Parametricity
- Dependent types
- Asynchrony
- Authorisation
- LNL and Higher-Order processes
- Logical Relations
- Encoding Multiparty Systems
- Non-determinism (forthcoming)

# Core References

Caires, Pfenning: Session Types as Intuitionistic Linear Propositions. **CONCUR 10**

Toninho, Caires, Pfenning: Dependent session types via intuitionistic linear type theory. **PPDP 11**

Caires, Pfenning, Toninho: Towards concurrent type theory. **TLDI 12**

Toninho, Caires, Pfenning: Functions as Session-Typed Processes. **FoSSaCS 12**

Pérez, Caires, Pfenning, Toninho: Linear Logical Relations for Session-Based Concurrency. **ESOP 12**

DeYoung, Caires, Pfenning, Toninho: Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication. **CSL 12**

Wadler: Propositions as sessions. **ICFP 12** **(**also **JFP 14)**

Toninho, Caires, Pfenning: Higher-Order Processes, Functions, and Sessions: A Monadic Integration. **ESOP 13**

Caires, Pérez, Pfenning, Toninho: Behavioral Polymorphism and Parametricity in Session-Based Communication. **ESOP 13**

Toninho, Caires, Pfenning: Corecursion and Non-divergence in Session-Typed Processes. **TGC 14**

Caires, Pfenning:, Toninho, Linear Logic Propositions as Session Types. **MSCS 16**

Caires, Pérez: Multiparty Session Types Within a Canonical Binary Theory, and Beyond. **FORTE 16**

# Background

Wadler: Propositions as types. Commun. ACM 58(12) (2015)

Cardelli: Typeful Programming, IFIP State-of-the-Art Reports (1989)

Milner, Parrow, Walker: A Calculus of Mobile Processes, I. Inf. Comput. 100(1): 1-40 (1992)

Milner: Functions as Processes. Mathematical Structures in Computer Science 2(2): (1992)

Gay: A Sort Inference Algorithm for the Polyadic Pi-Calculus. POPL 1993

Pierce, Sangiorgi: Behavioral equivalence in the polymorphic pi-calculus. J. ACM 47(3): (2000)

Pierce, Sangiorgi: Typing and Subtyping for Mobile Processes. Mathematical Structures in Computer Science 6(5) (1996)

Merro, Sangiorgi: On Asynchrony in Name-Passing Calculi. ICALP 1998

Sangiorgi: The Name Discipline of Uniform Receptiveness. ICALP 1997

Kobayashi, Pierce, Turner: Linearity and the pi-calculus. ACM Trans. Program. Lang. Syst. 21(5): 7 (1999)

Honda: Types for Dyadic Interaction. CONCUR 1993

Honda, Vasconcelos, Kubo: Language Primitives and Type Discipline for Structured Communication-Based Programming. ESOP 1998

Gay, Hole: Subtyping for session types in the pi calculus. Acta Inf. 42(2-3) (2005)

Giunti, Vasconcelos: A Linear Account of Session Types in the Pi Calculus. CONCUR 2010

## Background

Honda, Laurent: An exact correspondence between a typed pi-calculus and polarised proof-nets. Theor. Comput. Sci. 411(22-24): (2010)

Bellin, Scott: On the pi-Calculus and Linear Logic. Theor. Comput. Sci. 135(1): (1994)

Abramsky: Computational Interpretations of Linear Logic. Theor. Comput. Sci. 111(1&2): (1993)

Andreoli: Logic Programming with Focusing Proofs in Linear Logic. J. Log. Comput. 2(3): 347 (1992)

Barber, Plotkin: Dual Intuitionistic Linear Logic, ECS-LFCS-96-347, 1996.

Benton: A Mixed Linear and Non-Linear Logic: Proofs, Terms and Models. CSL 1994