DRAFT 3.5

LINEAR LOGIC AND BEHAVIORAL TYPES (1)

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

NOVALINCS
NOVA Laboratory for
Computer Science and Informatics

BETTY 2016 Limassol Cyprus

type systems for programming

- Types at the heart of common 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!
- Impact in mainstream technology
  - "standard" types must be easy to use by any programmer

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, see e.g,, Rust).
  - We expect behavioural types will lead to a new generation of type systems for future programming languages

simply typed λ-calculus [Church30]

$\Gamma \vdash M{:}\mathrm{A}$

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

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

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

simply typed λ-calculus

$$\Gamma \vdash * : 1$$

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

$$\frac{\Gamma \vdash M : \mathrm{A} \wedge \mathrm{B}}{\Gamma \vdash \mathsf{fst}(M) : \mathrm{A}}$$

$$\mathtt{Tfst}(\mathtt{Tpair}(d_1, d_2)) \to d_1$$

$$\frac{\Gamma \vdash M : \mathrm{A} \wedge \mathrm{B}}{\Gamma \vdash \mathsf{snd}(M) : \mathrm{B}}$$

$$\mathtt{Tsnd}(\mathtt{Tpair}(d_1, d_2)) \to d_2$$

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

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

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

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

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

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

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

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

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

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

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
- "***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]

Types for Processes

the π-calculus [Milner92]

$$
\begin{array}{rlll}
P & ::= & \mathbf{0} & \text{(inaction)} \\
 & | & P \mid Q & \text{(composition)} \\
 & | & (\mathsf{new}\ x)P & \text{(restriction)} \\
 & | & x(y).P & \text{(input)} \\
 & | & x[y].P & \text{(free output)} \\
 & | & !x(y).P & \text{(replicated input)} \\
 & | & \bar{x}(y).P & := (\mathsf{new}\ y)x[y].P \quad \text{(fresh output)}
\end{array}
$$

Semantics:

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

the π-calculus [Milner92]

structural congruence (≡)

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

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

the π-calculus [Milner92]

reduction ($\rightarrow$)

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

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

Types for Processes

simple types [Milner92,Gay93]

simple types [Milner92,Gay93]

U ::= bool (base type)
| [U] (send/receive of U)

$$\Gamma \vdash 0 \qquad \frac{\Gamma, x{:}[\mathrm{U}] \vdash \mathrm{P}}{\Gamma \vdash (\mathsf{new}\ x)\mathrm{P}} \qquad \frac{\Gamma \vdash \mathrm{P} \quad \Gamma \vdash \mathrm{Q}}{\Gamma \vdash \mathrm{P} \mid \mathrm{Q}}$$

$$\frac{\Gamma \vdash \mathrm{P}}{\Gamma \vdash !\mathrm{P}} \qquad \frac{\Gamma, a{:}[\mathrm{U}] \vdash v : \mathrm{U}}{\Gamma, a{:}[\mathrm{U}] \vdash a[v].\mathrm{P}} \qquad \frac{\Gamma, a{:}[\mathrm{U}], x{:}\mathrm{U} \vdash \mathrm{P}}{\Gamma, a{:}[\mathrm{U}] \vdash a(x).\mathrm{P}}$$

$$\frac{\Gamma \vdash b{:}\mathrm{bool} \quad \Gamma \vdash \mathrm{P} \quad \Gamma \vdash \mathrm{Q}}{\Gamma \vdash \mathtt{if}\ b\ \mathtt{then}\ \mathrm{P}\ \mathtt{else}\ \mathrm{Q}}$$

IO-types [PierceSangiorgi93]

IO-types [PierceSangiorgi93]

Typing and Subtyping for Mobile Processes

Benjamin Pierce* Davide Sangiorgi†

May 10, 1994

Abstract

The π-calculus is a process algebra that supports process mobility by focusing on the communication of channels. Milner's presentation of the π-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 λ-calculi.

IO-types [PierceSangiorgi93]

U ::=	bool	(base type)
	| [U]	(receive/send of U)
	| ?U	(receive of U)
	| !U	(send of U)

$$\frac{\Gamma, a{:}?\mathrm{B}, x{:}\mathrm{B} \vdash \mathrm{P}}{\Gamma, a{:}?\mathrm{B} \vdash a(x).\mathrm{P}}$$

$$\frac{\Gamma, a{:}!\mathrm{B} \vdash v : \mathrm{B}}{\Gamma, a{:}!\mathrm{B} \vdash a[\mathrm{v}].\mathrm{P}}$$

Linear Types [KobayashiPierceTurner96]

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

Linear Types [KobayashiPierceTurner96]

U ::= bool (base type)
| ?U (input of U)
| !U (output of U)
| !?U (i/o of U)
| *?U (input of U)
| *!U (output of U)
| *!?U (output of U)

- Δ linear typing context (multiset)
- Γ cartesian typing context (set)

$$\Gamma; \Delta \vdash P$$

$$\Gamma; \cdot \vdash 0 \qquad \frac{\Gamma; \Delta_1 \vdash P \quad \Gamma; \Delta_2 \vdash Q}{\Gamma; \Delta_1, \Delta_2 \vdash P \mid Q}$$

$$\Delta, x{:}!\mathrm{U}, x{:}?\mathrm{U} = \Delta, x{:}!?\mathrm{U}$$

Linear Types [KobayashiPierceTurner96]

U ::= bool	(base type)
| ?U	(input of U)
| !U	(output of U)
| !?U	(i/o of U)
| *U	(shared)

$$\Gamma; \cdot \vdash 0 \qquad \frac{\Gamma; \Delta_1 \vdash \mathrm{P} \quad \Gamma; \Delta_2 \vdash \mathrm{Q}}{\Gamma; \Delta_1, \Delta_2 \vdash \mathrm{P} \mid \mathrm{Q}}$$

$$\Gamma; x{:}\mathrm{U} \vdash x{:}\mathrm{U} \qquad \frac{\Gamma; \Delta, x{:}\mathrm{U} \vdash \mathrm{P}}{\Gamma; \Delta \vdash (\mathsf{new}\ x)\mathrm{P}}$$

$$\frac{\Gamma;\Delta_1 \vdash x{:}!\mathrm{U} \quad \Gamma;\Delta_2 \vdash y{:}\mathrm{U} \quad \Gamma; \Delta_3 \vdash \mathrm{P}}{\Gamma; \Delta_1, \Delta_2, \Delta_3 \vdash x[y].\mathrm{P}} \qquad \frac{\Gamma; \Delta, y{:}\mathrm{U} \vdash \mathrm{P}}{\Gamma; \Delta, x{:}?\mathrm{U} \vdash x(\mathrm{y}).\mathrm{P}}$$

$$\Gamma, x{:}\mathrm{U}; \cdot \vdash x{:}{*}\mathrm{U} \qquad \frac{\Gamma, x{:}\mathrm{U}; \Delta \vdash \mathrm{P}}{\Gamma; \Delta, x{:}{*}\mathrm{U} \vdash \mathrm{P}} \qquad \frac{\Gamma, x{:}\mathrm{A}; y{:}\mathrm{A} \vdash \mathrm{P}}{\Gamma, x{:}\mathrm{A}; \cdot \vdash !x(y).\mathrm{P}}$$

Linear Types [KobayashiPierceTurner96]

U ::= bool	(base type)
| ?U	(input of U)
| !U	(output of U)
| !?U	(i/o of U)
| *U	(shared)

$$\Gamma;\cdot\vdash 0$$

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

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

$$\frac{\Gamma;\Delta, x{:}!U, x{:}?U\vdash P}{\Gamma;\Delta\vdash (\mathsf{new}\ x)P}$$

$$\frac{\Gamma;\Delta_1\vdash x{:}!U \quad \Gamma;\Delta_2\vdash y{:}U \quad \Gamma;\Delta_3\vdash P}{\Gamma;\Delta_1,\Delta_2,\Delta_3\vdash x[y].P}$$

$$\frac{\Gamma;\Delta, y{:}U\vdash P}{\Gamma;\Delta, x{:}?U\vdash x(y).P}$$

$$\Gamma, x{:}U;\cdot\vdash x{:}{*}U$$

$$\frac{\Gamma, x{:}U;\Delta\vdash P}{\Gamma;\Delta, x{:}{*}U\vdash P}$$

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

Session Types [Honda93,HKV98,GH05]

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 β-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 π-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]

$$T ::= {*T} \quad \text{(shared channel)} \mid S \quad \text{(session type)}$$

$$S ::= \mathsf{end} \quad \text{(base type)} \mid\ !T.S \quad \text{(output)} \mid\ ?T.S \quad \text{(input)}$$

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

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

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

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

Curry Howard for Process Types?

Computational Interpretations of LL

On the π-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 π-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 π-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 π-calculus).

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 β-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

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 π-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 π-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 π-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)
	|	1	(end)

Duality on session types, the key insight of [H93], captured by duality at the logical level.

Linear Sequent Calculus [Andreolli92]

$$\Gamma; \Delta \vdash A$$

- Δ linear context (multiset)
- Γ cartesian context (set)

$$\Gamma; A \vdash A$$

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

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

$$\frac{\Gamma; \Delta, A \vdash B}{\Gamma; \Delta \vdash A \multimap B}$$

$$\Gamma; - \vdash 1$$

$$\frac{\Gamma; \Delta_1 \vdash A \quad \Gamma; \Delta_2 \vdash B}{\Gamma; \Delta_1, \Delta_2 \vdash A \otimes B}$$

$$\frac{\Gamma; \Delta, A, B \vdash C}{\Gamma; \Delta, A \otimes B \vdash C}$$

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

Sequent calculus presentation of DILL [BarberPlotkin91]

Linear Propositions as Session Types

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

$$\frac{\Gamma; \Delta \vdash P :: C}{\Gamma; \Delta, y{:}1 \vdash P :: C} \qquad \Gamma; \vdash 0 :: y{:}1$$

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

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

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)

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]

Movie Server Session

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

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

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

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

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

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

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

Shared Movie Server

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

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

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

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

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

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

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

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

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

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

$$\rightarrow$$

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

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

Send and Receive

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

$$\rightarrow$$

Send and Receive

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

$$\rightarrow$$

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

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

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

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

$$\rightarrow$$

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

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

Choice and Offer

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

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

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

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

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

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

Choice and Offer

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

$$\rightarrow$$

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

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

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

Admissible Rules (DILL)

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

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

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

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

Copycat Forwarder

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

$$\rightarrow$$

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

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

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

Duality in DILL

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

$$\overline{U \otimes S} = U \multimap \overline{S}$$
$$\overline{U \multimap S} = U \otimes \overline{S}$$
$$\overline{S \oplus S} = \overline{S} \& \overline{S}$$
$$\overline{S \& S} = \overline{S} \oplus \overline{S}$$
$$\overline{1} = 1$$

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

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 ($\equiv$)

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

Proof Reductions = Process Reductions

Computational Conversions ($\to$)

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

Correspond to standard process reductions ($\to$)

$$(\mathsf{new}\ x)(x(y).P \mid \bar{x}(y).(Q \mid R)) \ \to (\mathsf{new}\ x)(P \mid (\mathsf{new}\ y)(Q \mid \mathrm{R}))$$
$$(\mathsf{new}\ x)(x.\mathtt{case}(Q,P) \mid x.\mathtt{inl};R) \to (\mathsf{new}\ x)(Q \mid R)$$
$$(\mathsf{new}\ x)(!x(y).P \mid \bar{x}(y).Q) \to (\mathsf{new}\ y)(P \mid (\mathsf{new}\ x)(!x(y).P \mid Q))$$

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]
- Yet another remarkable bridge surfacing here

Proof Conversions = Process Identities

- Structural Conversions ($\equiv$)

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

- Computational Conversions ($\longrightarrow$)

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

- Structural Conversions ($\simeq$)

  ($\simeq$) matched by typed π 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 $P \rightarrow Q$

LINEAR LOGIC AND BEHAVIORAL TYPES (2)

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

NOVALINCS

NOVA Laboratory for
Computer Science and Informatics

BETTY 2016 Limassol Cyprus

Session Types as Linear Propositions

$$
\begin{array}{lll}
S,U ::= & & \\
\quad \mid & U \otimes S & \text{(output)} \\
\quad \mid & U \multimap S & \text{(input)} \\
\quad \mid & S \oplus S & \text{(choice)} \\
\quad \mid & S \& S & \text{(offer)} \\
\quad \mid & !U & \text{(shared)} \\
\quad \mid & 1 & \text{(end)}
\end{array}
$$

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)
	$\mid$	$!x(y).P$	(replicated server)
	$\mid$	$x.\mathtt{case}(P,Q)$	(offer)
	$\mid$	$x.\mathtt{inl};Q$	(choose left)
	$\mid$	$x.\mathtt{inr};Q$	(choose right)

Linear Propositions as Session Types

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

$$\frac{\Gamma; \Delta \vdash P :: C}{\Gamma; \Delta, y{:}1 \vdash P :: C} \qquad \Gamma; \vdash 0 :: y{:}1$$

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

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

Curry-Howard Correspondence

Theorem (*processes as proofs*)

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)*

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*)

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

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

Coming up next

- Some Examples
- Sharing and Duality
- Systems based on Classical Linear Logic

  Sharing, Locality and Receptiveness
- Behavioural Polymorphism

  Logical Relations and Parametricity

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 ⇆ 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

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

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

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

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

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

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

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

Movie Server Session

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

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

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

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

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

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

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

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

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

Replication and Sharing

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

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

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

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

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

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

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

Replication and Sharing

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

$$\equiv$$

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

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

Replication and Sharing

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

$$\rightarrow$$

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

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

Shared Movie Server

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

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

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

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

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

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

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

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

Shared Movie Server

- ; - $\vdash$ (new srv)($Mov(srv)$ | $SA(srv)$ | $SB(srv)$)

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

- ; - $\vdash$ (new srv)($Mov(srv)$ | $SA(srv)$) | (new srv)($Mov(srv)$ | $SB(srv)$)

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

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

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

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

$\longrightarrow^*$

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

DILL and Locality

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

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

- !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:

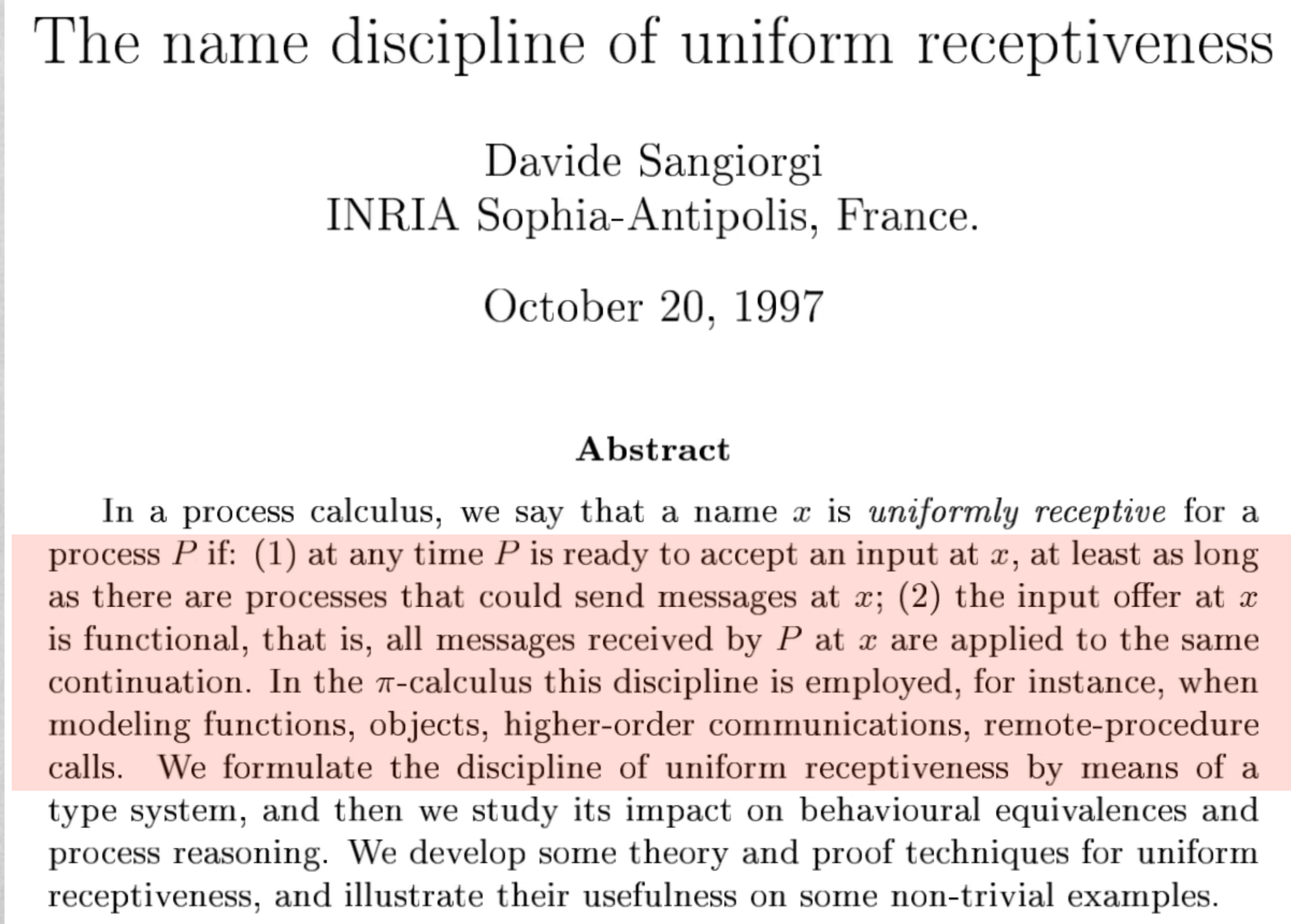
  $!A \multimap B$

- 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 π-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 π-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* π (Lπ), 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 Lπ. We focus on bisimulation-based behavioural equivalences, precisely on *barbed congruence*. We prove two coinductive characterisations of barbed congruence in Lπ, 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 λ-calculus; the validity of some laws for Join.

Duality for All Session Types

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

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

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

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

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

$$\overline{1} = 1$$

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

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

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

Session Types as CLL Propositions

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

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

$$\overline{\overline{S}} = S$$
$$\overline{S} = S^{\perp}$$
$$U \multimap S = \overline{U} ⅋ S$$

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 π-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 π-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]

$$\vdash \Delta; \Theta$$

- Δ linear context (multiset)
- Θ cartesian context (set)

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

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

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

NB. This system corresponds to a classical version of DILL

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

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

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

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

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

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

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

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

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

Classical Linear Logic [TCP'12-14]

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

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

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

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

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

$$\longrightarrow$$

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

Proofs = Processes

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

Proof Conversions = Process Identities

- Structural Conversions ($\equiv$)

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

- Computational Conversions ($\longrightarrow$)

  ($\longrightarrow$) matched by π reduction ($\longrightarrow$)

- 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 ($\longrightarrow$)

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

correspond to standard process reductions ($\to$)

$(\mathsf{new}\, x)(x.\overline{\mathtt{close}} \mid x.\mathtt{close}.P) \;\to P$

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

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

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

Proof Conversions = Process Identities

- Structural Conversions (≃)

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

$$(\mathsf{new}\ x)(!x(y).P \mid (\mathsf{new}\ z)(Q \mid R)) \simeq$$
$$\quad (\mathsf{new}\ z)(\ (\mathsf{new}\ x)(!x(y).P \mid Q) \mid (\mathsf{new}\ x)(!x(y).P \mid R))$$
$$(\mathsf{new}\ x)(!x(y).P \mid (\mathsf{new}\ z)(!z(u).Q \mid R)) \simeq$$
$$\quad (\mathsf{new}\ z)(\ !z(u).(\mathsf{new}\ x)(!x(y).P \mid Q) \mid (\mathsf{new}\ x)(!x(y).P \mid R))$$
$$(\mathsf{new}\ x)(!x(y).P \mid Q) \simeq Q\ \ [\ x \notin fn(Q)\ ]$$

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

Proof Conversions = Process Identities

- Structural Conversions (≃)

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

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

cut/mix conversions

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

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

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

CLL is free from locality

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

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

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

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

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

- 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

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

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

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

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

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

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

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]

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

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

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

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

Polymorphic λ-calculus [Girard-Reynolds]

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

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

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

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

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

Polymorphic λ-calculus [Girard-Reynolds]

$$\frac{\Omega;\Gamma \vdash M\!: B\{S/X\} \quad \Omega \vdash S\ \mathbf{ty}}{\Omega;\Gamma \vdash \langle S,M\rangle : \exists X.B}$$

$$\frac{\Omega;\Gamma \vdash M\!: \exists X.B \quad \Omega,X;,\ x{:}X \vdash N{:}A}{\Omega;\Gamma \vdash \texttt{let}\ \langle X,x\rangle = M\ \texttt{in}\ N{:}A}$$

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

Linear Propositions as Session Types

- Typing judgement

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

- Intuition: judgement states a rely-guarantee property:

for all session types Ω, whenever composed with processes offering a session A_i at x_n, *P* offers a session of type C at y

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

typing ensures fidelity and global progress (cut-elimination)

Proofs = Processes

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

Linear Propositions as Session Types

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

$$\frac{\Omega, X;\Gamma;\Delta, x{:}B \vdash P :: C}{\Omega;\Gamma;\Delta, x{:}\exists X.B \vdash x(X).P :: C}$$

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

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

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 to the Cloud

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

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

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

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

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

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

Creating a Custom Service

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

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

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

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

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

Creating a Custom Service

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

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

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

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

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

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

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

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

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

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

$P \in T_\eta^\omega[\![z{:}1]\!] \triangleq \forall Q.\ (P \Rightarrow Q \wedge Q \not\rightarrow\ ) \supset Q \equiv_! \mathbf{0}$

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

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

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

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

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

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

Logical Candidate

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

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

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

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

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

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

- 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".

LINEAR LOGIC AND BEHAVIORAL TYPES (3)

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

NOVALINCS
NOVA Laboratory for Computer Science and Informatics

BETTY 2016 Limassol Cyprus

Representing Data Types

Typeful Encodings of Data

- In Milner-style encodings of data as processes [Milner89] a value V is represented by a **process** $[\![\mathbf{V}]\!]_n$ located at name n.
- $[\![V]\!]_n$ accessed "by reference" through the unique "address" n
- [Milner91] showed how to embed the λ-calculus in the π-calculus just by using name passing
- usage of n n may be linear (e.g., V is resource, e.g., a lock or a continuation) or shared (V is a value, e.g., a function, a bool)
- Curry-Howard typing for sessions promotes "free" **typeful constructions** of higher-order data types "as processes", in the style of constructive type theory.

Typeful Encodings of Data

$\mathit{LinearBool} \triangleq \mathtt{1} \oplus \mathtt{1}$

$\mathit{LinearTrue}(s) \triangleq s.\mathtt{inr};0 \qquad \mathit{LinearFalse}(s) \triangleq s.\mathtt{inl};0$

$\mathit{Bool} \triangleq \,!\mathit{LinearBool}$

$\mathit{True}(s) \triangleq \,!b(s)\mathit{LinearTrue}(s) \qquad \mathit{False}(s) \triangleq \,!b(s)\mathit{LinearFalse}(s)$

$\mathtt{if}(b, P, Q) \triangleq \overline{b}(c).\mathrm{c.case}(P, Q)$

$\Gamma; \Delta_1 \vdash P :: \mathrm{C}$

$\Gamma; \Delta_2 \vdash Q :: \mathrm{C}$

$\Gamma; b{:}\mathit{Bool}, \Delta_1, \Delta_2 \vdash \mathtt{if}(b, P, Q) :: \mathrm{C}$

Linear Pairs

$LPair(\mathrm{A}_1, \mathrm{A}_2) \triangleq \mathrm{A}_1 \otimes \mathrm{A}_2 \otimes 1$ $[\![\mathrm{V}]\!]_\mathrm{n}$

$\Gamma;\ \Delta_1 \vdash [\![\mathrm{V}_1]\!]_z :: z{:}\mathrm{A}_1$

$[\![<\mathrm{V}_1, \mathrm{V}_2>\ ]\!]_s \triangleq \overline{s}(x).([\![\mathrm{V}_1]\!]_x \mid \overline{\mathrm{s}}(y).([\![\mathrm{V}_2]\!]_y \mid \mathbf{0}))$

$\Gamma;\ \Delta_1, \Delta_2 \vdash [\![<\mathrm{V}_1, \mathrm{V}_2>\ ]\!]_s :: s{:}LPair(A_1, A_2)$

$[\![\ \mathtt{let}\ (x,y) = \mathrm{V}_1\ \mathtt{in}\ P\ ]\!]_s \triangleq (\mathtt{new}\ p)(\ [\![\mathrm{V}_1]\!]_p \mid p(x).p(y).\ [\![\mathrm{V}_2]\!]_z)$

Pure Pairs

$Pair(A_1,A_2) \triangleq\ !(!A_1 \otimes !A_2 \otimes 1)$ $\qquad\qquad$ $[\![V]\!]_n$

$\Gamma;\ \Delta_1 \vdash [\![V_1]\!]_z :: z{:}!A_1$

$[\![<V_1, V_2>\ ]\!]_s \triangleq\ !s(p).\overline{s}(x).([\![V_1]\!]_x \mid \overline{s}(y).([\![V_2]\!]_y \mid \mathbf{0}))$

$\Gamma;\ \Delta_1, \Delta_2 \vdash [\![<V_1, V_2>\ ]\!]_s :: s{:}Pair(A_1,A_2)$

$[\![\ \mathtt{fst}(V)\ ]\!]_k \triangleq (\mathsf{new}\ s)(\ [\![V]\!]_s \mid \bar{s}(p).p(x).p(y).[x \leftrightarrow k])$

$[\![\ \mathtt{snd}(V)\ ]\!]_k \triangleq (\mathsf{new}\ s)(\ [\![V]\!]_s \mid \bar{s}(p).p(x).p(y).[y \leftrightarrow k])$

Linear λ-calculus [BarberPlotkin96]

$$\Gamma; \Delta \vdash A$$

- Δ linear context (multiset)
- Γ exponential context (set)

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

$$\frac{\Gamma;\Delta, x{:}A \vdash M{:}B}{\Gamma; \Delta \vdash \lambda x{:}A.M{:}A \multimap B}$$

$$\frac{\Gamma; \Delta_1 \vdash M{:}A \multimap B \quad \Gamma; \Delta_2 \vdash N{:}A}{\Gamma; \Delta_1, \Delta_2 \vdash MN{:}B}$$

$$\frac{\Gamma; \Delta_1 \vdash M{:}A \qquad \Gamma;\Delta_2 \vdash N{:}B}{\Gamma; \Delta_1, \Delta_2 \vdash \langle M,N\rangle{:}A \otimes B}$$

$$\frac{\Gamma; \Delta_1 \vdash M{:}A \otimes B \quad \Gamma; \Delta_2, x{:}A, y{:}B \vdash N{:}C}{\Gamma; \Delta_1, \Delta_2 \vdash \mathtt{let}\ \langle x,y\rangle = M\ \mathtt{in}\ N{:}C}$$

$$\frac{\Gamma; - \vdash M{:}A}{\Gamma; - \vdash M{:}!A}$$

$$\frac{\Gamma; \Delta_1 \vdash M{:}!A \quad \Gamma, x{:}A; \Delta_2 \vdash N{:}C}{\Gamma; \Delta_1, \Delta_2 \vdash \mathtt{let}\ !x = M\ \mathtt{in}\ N{:}C}$$

Linear λ-calculus [TCP12]

$[\![\Gamma; x{:}\mathrm{S} \vdash x{:}\mathrm{S}]\!]_z \triangleq [\![\Gamma; x{:}[\![\mathrm{S}]\!] \vdash [x{\leftrightarrow}z] :: z{:}[\![\mathrm{S}]\!]$

$[\![\Gamma, a{:}\mathrm{S}; \Delta \vdash a{:}!\mathrm{S}]\!]_z \triangleq [\![\Gamma, a{:}\mathrm{S}]\!]; [\![\Delta]\!] \vdash \overline{a}(x).[x{\leftrightarrow}z] :: z{:}![\![\mathrm{S}]\!]$

$[\![\Gamma; \Delta \vdash \lambda x{:}\mathrm{A}.M{:}\mathrm{S}{\multimap}\mathrm{U}]\!]_z \triangleq [\![\Gamma]\!]; [\![\Delta]\!] \vdash z(x)[\![M]\!]_z :: z{:}[\![\mathrm{S}]\!]{\multimap}[\![\mathrm{U}]\!]$

$[\![\Gamma; \Delta_1, \Delta_2 \vdash (MN){:}\mathrm{U}]\!]_z \triangleq$

$\quad [\![\Gamma]\!]; [\![\Delta_1, \Delta_2]\!] \vdash (\mathsf{new}\ \mathrm{s})([\![M]\!]_s \mid \overline{s}(h).([\![N]\!]_\mathrm{h} \mid [s{\leftrightarrow}z]) :: z{:}[\![\mathrm{U}]\!]$

$[\![\Gamma; - \vdash !\mathrm{M}{:}!\mathrm{S}]\!]_z \triangleq [\![\Gamma; - \vdash !z(x).[\![M]\!]_x :: z{:}![\![\mathrm{S}]\!]$

$[\![\Gamma; - \vdash \mathtt{let}\ !x{=}\mathrm{M}\ \mathtt{in}\ \mathrm{N}{:}\mathrm{S}]\!]_z \triangleq [\![\Gamma; - \vdash (\mathsf{new}\ x)([\![M]\!]_x \mid [\![N]\!]_\mathrm{z}) :: z{:}[\![\mathrm{S}]\!]$

Composing Translations [TCP12]

- Building on the translation of the λ-calculus into the session calculus, we mechanically extract translations of the pure λ-calculus into our session calculus
- The "canonical" translation

$$[\![S \to U]\!] \triangleq ![\![S]\!] \multimap [\![U]\!]$$

$$[\![t]\!] \triangleq t$$

One obtains Milner's **CBN** encoding

- The "boring" (following Girard) translation

$$[\![S \to U]\!] \triangleq !([\![S]\!] \multimap [\![U]\!])$$

$$[\![t]\!] \triangleq !t$$

One obtains **sharing evaluation**, cf. "futures" of Multilisp or Scala

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 our Curry-Howard mindset 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.

Mixed linear-non-linear Logic [Benton]

$\Gamma \vdash M : A$

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

- Δ linear channel context (multiset)
- Γ cartesian channel context (set)
- Ψ cartesian value context (set)

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

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

Mixed linear-non-linear Logic [Benton]

$\Gamma \vdash M : \mathrm{A}$

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

- Δ linear channel context (multiset)
- Γ cartesian channel context (set)
- Ψ cartesian value context (set)

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

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

Certifying Session Interfaces

"Standard" Session Type (talks about behaviour)

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

Dependent Session Type (talks about behaviour + data exchanged)

$$BankCI \triangleq \&\{\, \texttt{with}\colon \exists b{:}\mathbf{nat}.\ \forall v{:}\mathbf{nat}. \forall p{:}\ [v \leq b].\ \&\{\, \mathsf{ok};1, \mathsf{ko};1\},$$
$$\texttt{deposit}\colon \forall v{:}\mathbf{nat}.\ \forall p{:}\ [0 \leq v].\ \&\{\, \mathsf{ok};1, \mathsf{ko};1\}\}$$

Dependent Session Types

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

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

Certifying Session Interfaces

$$BankCI \triangleq \&\{\ \texttt{with:}\ \exists b{:}\mathbf{nat}.\ \forall v{:}\mathbf{nat}.\forall p{:}\ [v \leq b].\ \&\{\ \texttt{ok};1, \texttt{ko};1\},$$
$$\texttt{deposit:}\forall v{:}\mathbf{nat}.\ \forall p{:}\ [0 < v].\ \&\{\ \texttt{ok};1, \texttt{ko};1\}\}$$

$$Client(b) \triangleq b.\texttt{with}.\mathrm{s}(bv).\mathrm{s}(bv/2).\mathrm{s}[\mathrm{ltehalf}(bv)].\texttt{ok};1$$

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

Ψ contains a binding for ltehalf: $\forall b{:}\mathbf{nat}.\ \mathrm{b}/2 \leq b$

Mixed linear-non-linear Logic [Benton]

$\Gamma \vdash M : \mathrm{A}$

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

- Δ linear channel context (multiset)
- Γ cartesian channel context (set)
- Ψ cartesian value context (set)

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

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

App Store

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

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

A toy App Store

$AppStore \triangleq \&\{\ \texttt{game}\colon \{\ g\colon API \multimap Game\},$

$\texttt{maps}\colon \{\ g\colon API \multimap GPS \multimap Maps\ \}$

$\texttt{cam}\colon \{\ g\colon API \multimap CAM \multimap Cam\}\ \}$

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

$Betty(as,gps) \triangleq$

$as.\texttt{maps}.as(code).\texttt{spawn}\ g.\ (code\ \ulcorner g(api)\urcorner.g(gps).[g\leftrightarrow c]) \colon c{:}Maps$

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

The Cloud Server Type (redux)

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

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

Adding Recursion

- Both induction and replication allow for unbounded computation, but have quite different expressive power.
- E.g., we need both $!A$ and $\nu X.A$ in session types.
- Introducing general recursion in a logical system is challenging because we really require strong normalisation

Adding Recursion

Corecursion and Non-Divergence in Session Types

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

[1] Universidade Nova de Lisboa, Portugal
[2] Carnegie Mellon University, USA

Abstract. Session types are widely accepted as an expressive discipline for structuring communications in concurrent and distributed systems. In order to express infinitely unbounded sessions, session typed languages often include general recursion which may introduce undesirable divergence, e.g., infinite unobservable reduction sequences. In this paper we address, by means of typing, the challenge of ensuring non-divergence in a session-typed π-calculus with general (co)recursion, while still allowing interesting infinite behaviors to be definable. Our approach builds on a Curry-Howard correspondence between our type system and linear logic extended with co-inductive types, for which our non-divergence property implies consistency. We prove type safety for our framework, implying protocol compliance and global progress of well-typed processes. We also establish, using a logical relation argument, that well-typed processes are compositionally non-divergent, that is, that no well-typed composition of processes, including those dynamically assembled via name passing, can result in divergent behavior.

Coinductive Session Types

- Typing judgement

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

$$-;-;- \vdash (\mathsf{rec}\ X.y.\mathsf{case}(0, \bar{y}(-).X) :: x{:}\ \boldsymbol{\nu}Y.1\&(1\otimes Y)$$

- Key ideas: guardedness to enforce productivity, and co-regular recursion (other subtle conditions involved [CT14]).
- The assignment η keeps track of co-inductive assumptions associated to process variables

$$\eta(X(\boldsymbol{z})) = \Gamma;\Delta \vdash P :: x{:}Y$$

A Twitter Service

$\mathit{TrendService} \triangleq\ !(\mathit{Filter} \multimap \mathit{Trends})$

$\mathit{Filter} \triangleq\ !(\mathit{Tweets} \multimap \mathit{Trends})$

$\mathit{Tweets} \triangleq \boldsymbol{\nu}\mathrm{X}.(\mathbf{tweet} \otimes \mathrm{X})$

$\mathit{Trends} \triangleq \boldsymbol{\nu}\mathrm{X}.(\mathbf{trend} \otimes \mathrm{X})$

$\mathit{Client}(\mathrm{ts}) \triangleq \overline{\mathrm{ts}}(x).\overline{x}(\mathrm{f}).(\mathrm{AN}_{\mathrm{f}} \mid (\mathtt{rec}\ \mathrm{X}.x(t).\mathit{print}(t).\mathrm{X})$

$- ;\ ts{:}\mathit{TrendService} \vdash \mathit{Client}(\mathrm{ts}) :: \mathit{print} : \mathit{Trends}$

Logical Coinduction in Session Types

$$\frac{\eta';\Gamma;\Delta \vdash P :: x{:}A \quad \eta'=\eta[X(\boldsymbol{y}) / \Gamma;\Delta \vdash P :: x{:}Y]}{\eta; \Gamma; \Delta \vdash (\mathsf{rec}\ X(y).P\{y/z\})\ z :: x{:}\boldsymbol{\nu}Y.A}$$

$$\frac{\eta; \Gamma; \Delta, x{:}A\{X/\boldsymbol{\nu}X.A\} \vdash P :: C}{\eta;\Gamma;\Delta, x{:}\boldsymbol{\nu}X.A \vdash P :: C}$$

$$\frac{\eta(X(\boldsymbol{z})) = \Gamma;\Delta \vdash P :: x{:}Y}{\eta;\Gamma; \Delta \vdash X(\mathbf{z}) :: x{:}Y}$$

Logical Coinduction in Session Types

$$\frac{\eta';\Gamma;\Delta \vdash P :: x{:}A \quad \eta'=\eta[X(y)/\Gamma;\Delta \vdash P :: x{:}Y]}{\eta; \Gamma; \Delta \vdash (\mathsf{rec}\ X(y).P\{y/z\})\ z :: x{:}\boldsymbol{\nu}Y.A}$$

$$\equiv$$

$$\frac{\eta';\Gamma;\Delta \vdash P :: x{:}A \quad \eta'=\eta[X(y)/\Gamma;\Delta \vdash P :: x{:}Y]}{\eta; \Gamma; \Delta \vdash P\{(\mathsf{rec}\ X(y).P\{y/z\})/X\} :: x{:}A\{\boldsymbol{\nu}Y.AY\}}$$

Logical Coinduction in Session Types

$$\dfrac{\dfrac{\begin{array}{c}\eta' = \eta[X(\boldsymbol{y}) / \Gamma;\Delta \vdash P :: x{:}Y] \\ \eta';\Gamma;\Delta \vdash P :: x{:}A\end{array}}{\eta;\Gamma;\Delta \vdash (\mathsf{rec}\ X(\boldsymbol{y}).P\{\boldsymbol{y}/\boldsymbol{z}\})\ \boldsymbol{z} :: x{:}\boldsymbol{\nu}Y.A} \qquad \dfrac{\eta;\Gamma;\Delta, x{:}A\{X/\boldsymbol{\nu}X.A\} \vdash Q :: C}{\eta;\Gamma;\Delta, x{:}\boldsymbol{\nu}X.A \vdash Q :: C}}{\eta;\Gamma;\Delta \vdash (\mathsf{new}\ x)((\mathsf{rec}\ X(\boldsymbol{y}).P\{\boldsymbol{y}/\boldsymbol{z}\})\boldsymbol{z} \mid Q) :: C}$$

$$\equiv \longrightarrow$$

$$\dfrac{\Gamma;\Delta \vdash P\{(\mathsf{rec}\ X(\boldsymbol{y}).P\{\boldsymbol{y}/\boldsymbol{z}\})/X\} :: x{:}\boldsymbol{\nu}Y.A \quad \eta;\Gamma;\Delta, x{:}A\{X/\boldsymbol{\nu}X.A\} \vdash Q :: C}{\eta;\Gamma;\Delta \vdash (\mathsf{new}\ x)(P\{(\mathsf{rec}\ X(\boldsymbol{y}).P\{\boldsymbol{y}/\boldsymbol{z}\})/X\} \mid Q) :: C}$$

Key Result

Theorem
Let $\eta;\Gamma;\Delta \vdash P :: y{:}C$ be typable.
Then P is non-divergent (no infinite reduction).

Representing MultiParty Systems

Representing MultiParty Systems

- The linear logic typing discipline composes systems in pairs, through the duality matching expressed by the cut rule.
- Multiparty session types build on a notions of global types and projectability of global types into several local types, which are plain binary session types.
- Such projectability conditions typically ensure fidelity and (sometimes) progress (stuck freedom) of composed systems
- Linear logic gives an independent, yet equivalent, characterisation of the conditions isolated in [DenYos13] in their theory based on communicating automata.

Representing Multiparty Systems [CP14,16]

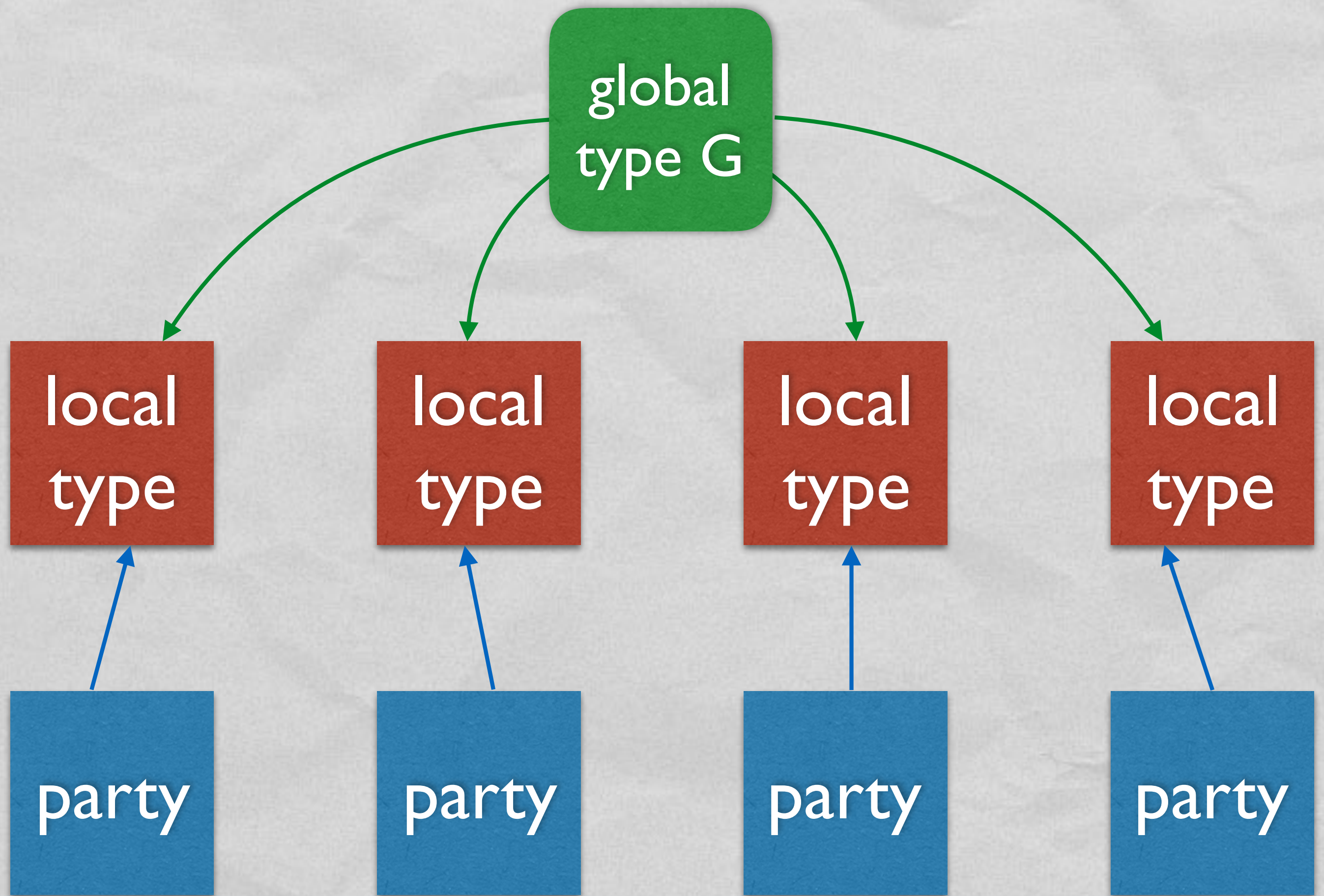
Multiparty Session Types Within A Canonical Binary Theory, and Beyond

Luís Caires[1] and Jorge A. Pérez[2]

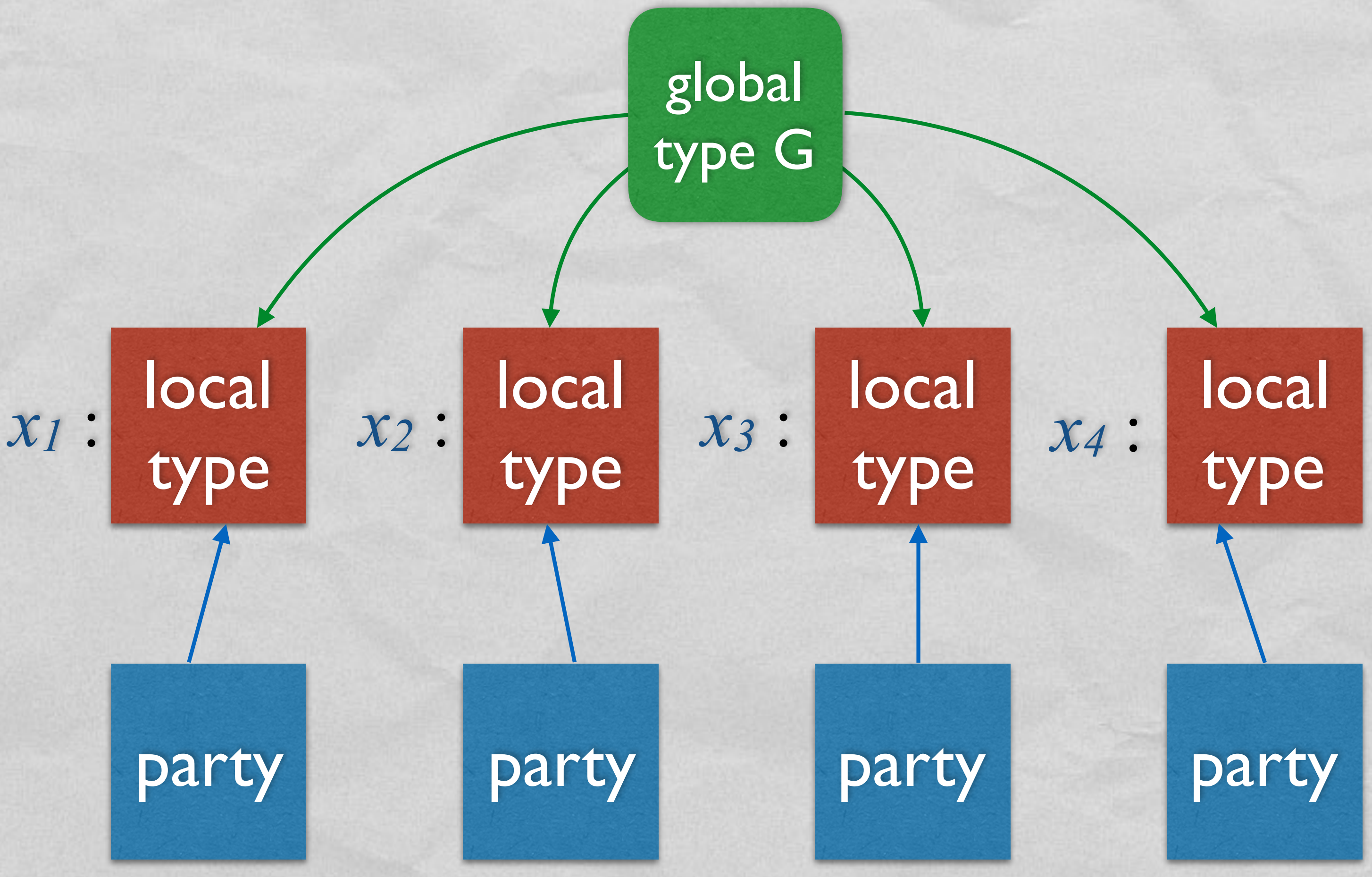
[1] NOVA LINCS - Universidade NOVA de Lisboa, Portugal
[2] University of Groningen, The Netherlands

Abstract. A widespread approach to software service analysis uses *session types*. Very different type theories for *binary* and *multiparty* protocols have been developed; establishing precise connections between them remains an open problem. We present the first formal relation between two existing theories of binary and multiparty session types: a binary system rooted in linear logic, and a multiparty system based on automata theory. Our results enable the analysis of multiparty protocols using a (much simpler) type theory for binary protocols, ensuring protocol fidelity and deadlock-freedom. As an application, we offer the first theory of multiparty session types with *behavioral genericity*. This theory is natural and powerful; its analysis techniques reuse results for binary session types.

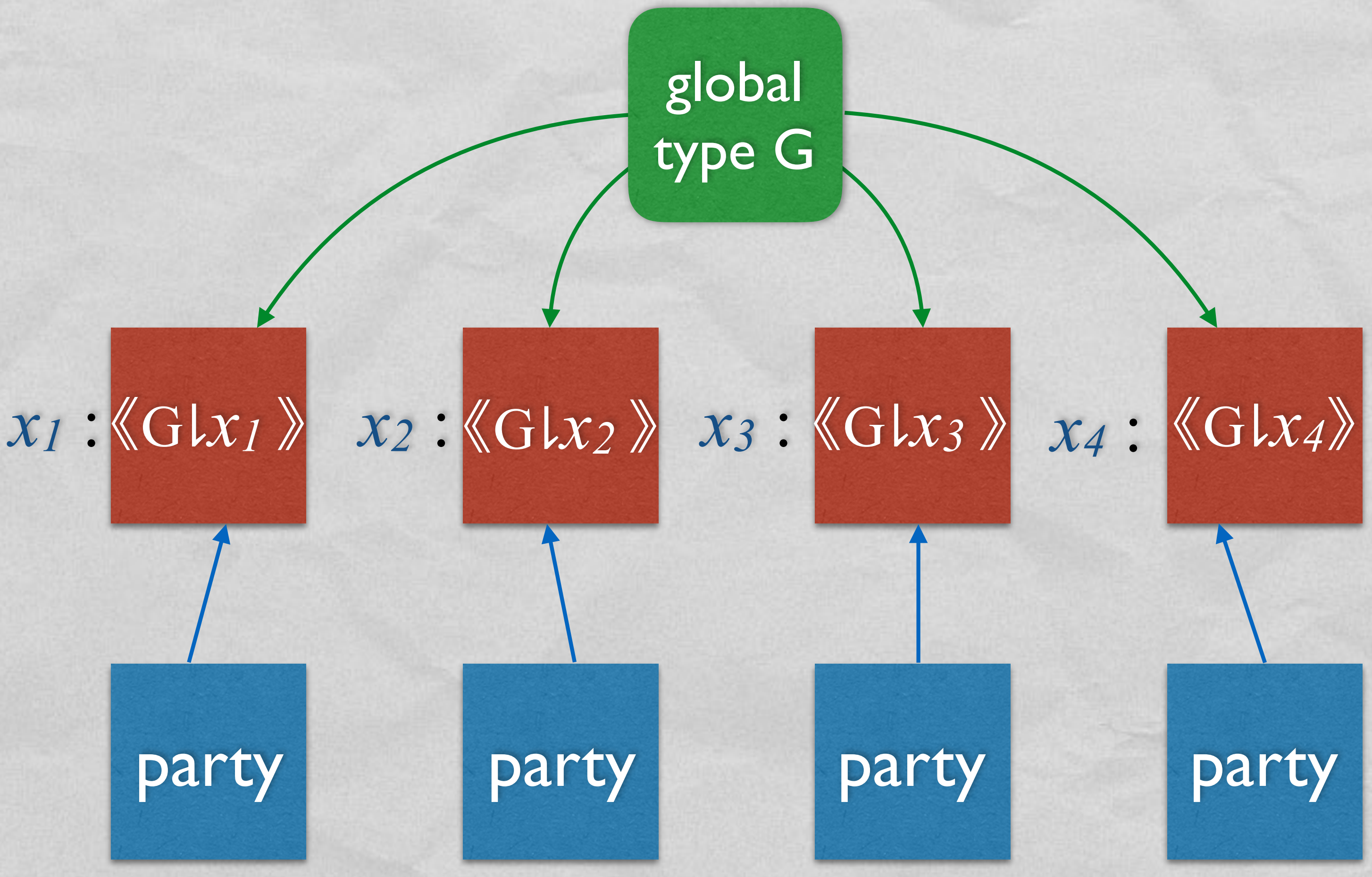
Key insight: Global Types as Medium Processes



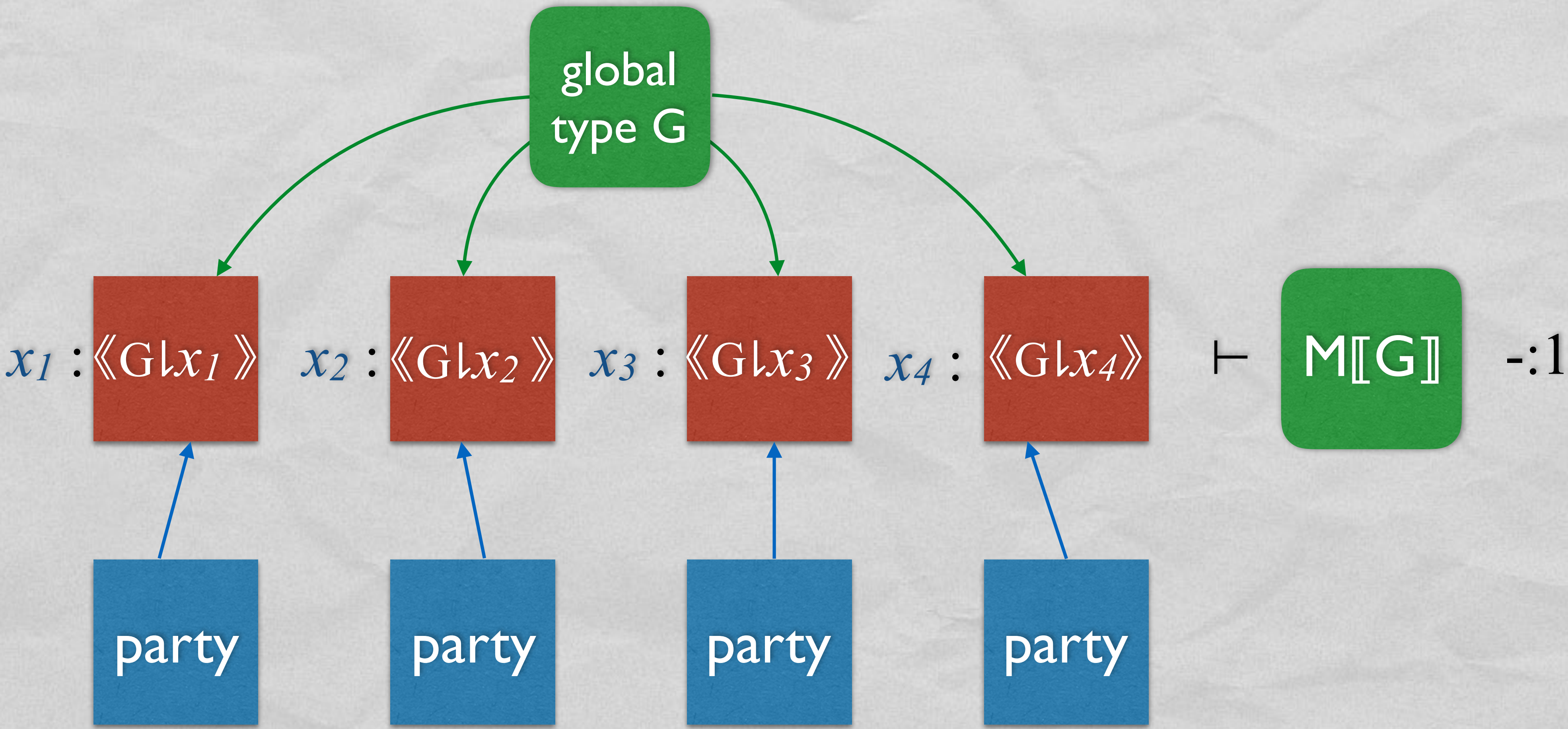
Key insight: Global Types as Medium Processes



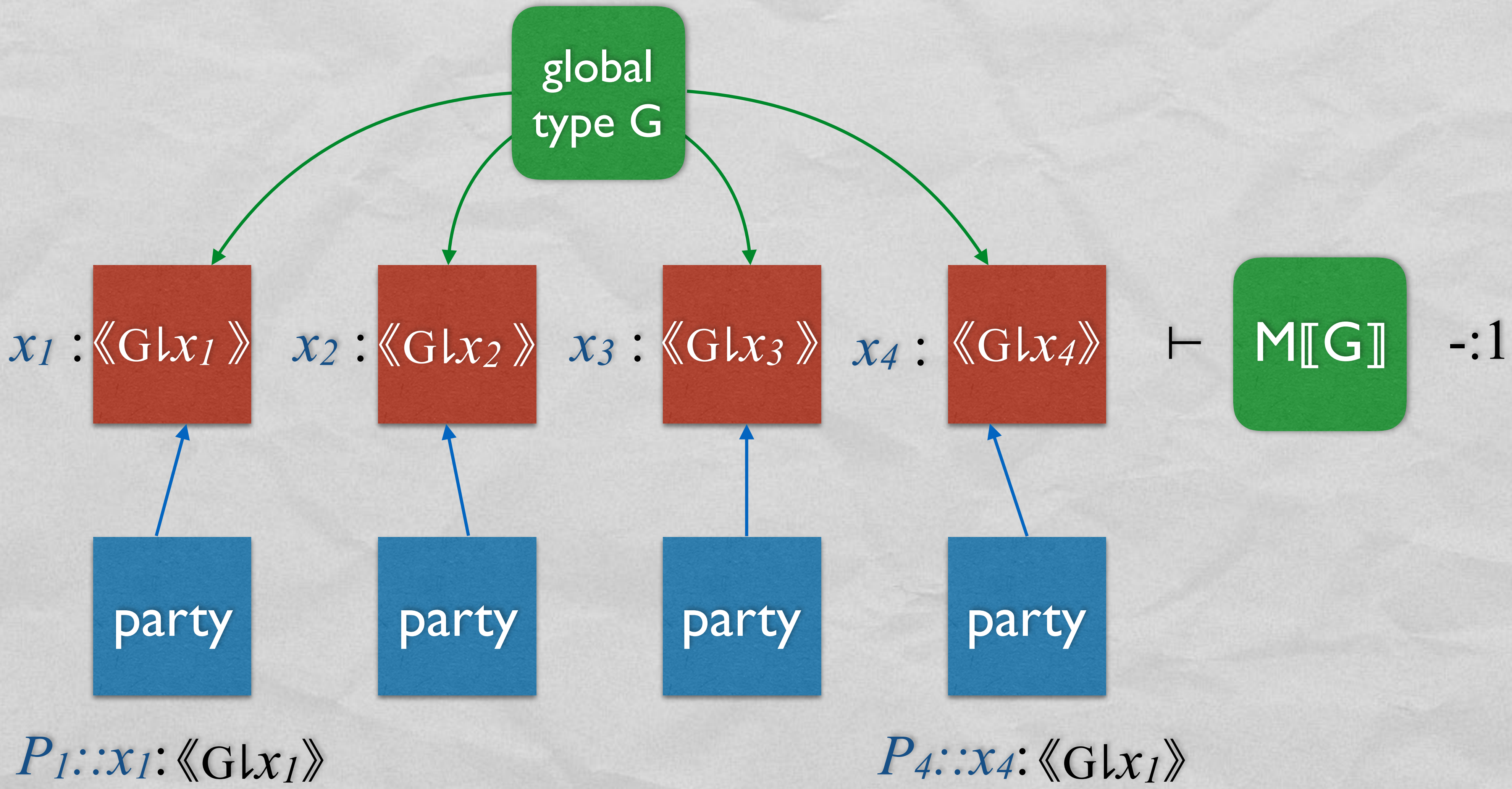
Key insight: Global Types as Medium Processes



Key insight: Global Types as Medium Processes

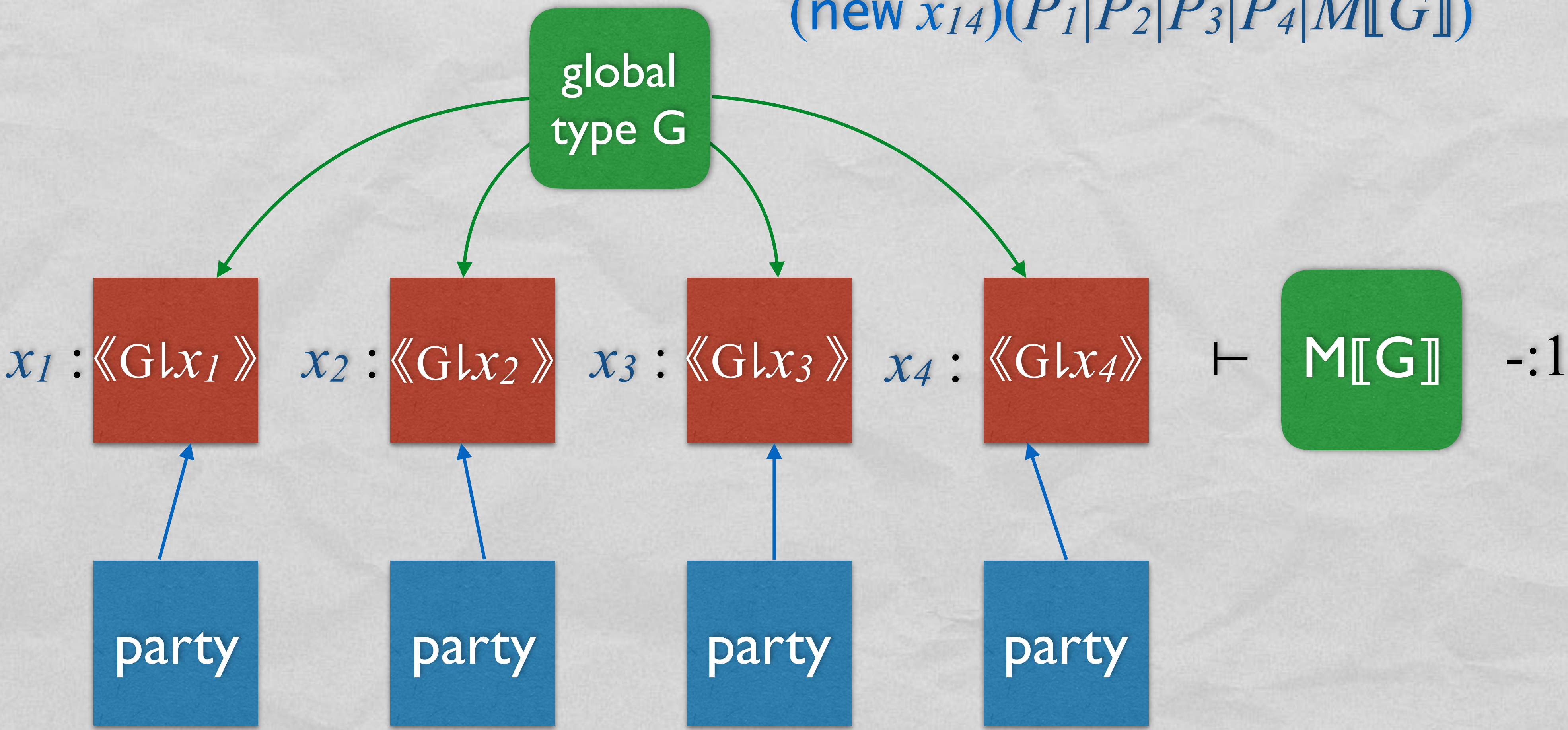


Key insight: Global Types as Medium Processes



Key insight: Global Types as Medium Processes

$(\text{new } x_{14})(P_1|P_2|P_3|P_4|M[\![G]\!])$



$P_1{::}x_1{:}\langle\!\langle G{\restriction}x_1 \rangle\!\rangle$

$P_4{::}x_4{:}\langle\!\langle G{\restriction}x_1 \rangle\!\rangle$

Projectability and Medium for Global Type

We consider a standard definition of projection [DY13] (ICALP)

$\langle\!\langle G \restriction p \rangle\!\rangle$ = … gives a standard session type

N.B. A global type G is well-formed if projectable in all parties

$\langle\!\langle T \rangle\!\rangle$ = … gives the linear logic session type corresponding to T

Medium for G:

$[\![\, p \to q : \{ \mathtt{l}_i[U_i].G_i \}_{i \in I}\ G]\!] \triangleq p.\mathtt{case}(\ldots \mathtt{l}_i : p(u).\ q.\mathtt{l}_i\,;\ q[u].\ [\![\, G_i \,]\!] \ldots)$

$[\![\, G_1 \mid G_2 \,]\!] \triangleq [\![\, G_1 \,]\!] \mid [\![\, G_2 \,]\!]$

$[\![\, \mathbf{end} \,]\!] \triangleq \mathbf{1}$

Characterization Results

Theorem (*global types as mediums*)
If G is a wf global type with part(G)=$\{x_1 \ldots x_n\}$ then
$\Gamma; x_1: \langle\!\langle G{\downharpoonleft}x_1 \rangle\!\rangle, \ldots, x_n: \langle\!\langle G{\downharpoonleft}x_n \rangle\!\rangle \vdash M[\![G]\!]$ *is typable*

Theorem (*mediums as global types)*
$\Gamma; x_1{:}A_1, \ldots, x_n{:}A_n \vdash M[\![G]\!]$ is typable then there are local types $T_1 \ldots T_n$ such than $\langle\!\langle T_i \rangle\!\rangle = A_i$ and $\langle\!\langle G{\downharpoonleft}x_i \rangle\!\rangle \sqsubseteq T_i$

Theorem (*operational correspondence)*
Let S = $(\mathsf{new}\ x_{1..n})(P_1 | \ \ldots\ | P_n | M[\![G]\!])$ be a system realizing G. Then the moves of S and G strongly agree.

Recent work by [CarLinMonSchWad16] build on our medium idea to relate several linear type systems for MPST based on a notion of multicut.

Summary

- Linearity plays a key role in logical and type systems for analysing the fine grained structure of proofs
- Linear logic offers a **complete Curry-Howard type theory** for name passing processes based on **session types**
- Strong properties obtained for free (e.g. global progress)
- This framework provides a sound basis for extensions, which often just come out quite naturally and harmoniously
- Many extensions, consequences and results have already been extracted from the basic framework.
- Many open questions around the corner …. !

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

[list under construction]