Mechanizing Session-Types: Enforcing linearity without linearity

Brigitte Pientka

McGill University
Montreal, Canada

beluga

joint work with
Chuta Sano and Ryan Kavanagh

Mechanizing formal systems together with proofs establishes trust.

Mechanizing formal systems together with proofs establishes trust... and avoids flaws.

Programs Go Wrong

Compilers Go Wrong

Compiler bugs

- are very hard to find and to fix
- every programmer and every program is affected and potentially at risk

  $\Longrightarrow$ over 195 bugs in GCC and LLVM C Compiler [Vu et al. PLDI'14]

Compilers Go Wrong

Compiler bugs

- are very hard to find and to fix
- every programmer and every program is affected and potentially at risk

  $\Longrightarrow$ over 195 bugs in GCC and LLVM C Compiler [Vu et al. PLDI'14]

Grand Challenge: Verifying compiler challenge (T. Hoare)

1970s — Challenge proposed, but abandoned

2003 — Challenge renewed

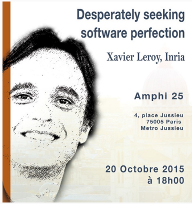
Compilers Go Wrong

Compiler bugs

- are very hard to find and to fix
- every programmer and every program is affected and potentially at risk
  $\Longrightarrow$ [Vu et al. PLDI'14] found over 195 bugs in GCC and LLVM C Compiler

Grand Challenge: Verifying compiler challenge (T. Hoare)

1970s	Challenge proposed, but abandoned
2003	Challenge renewed
2009	**CompCert Compiler** [X. Leroy]



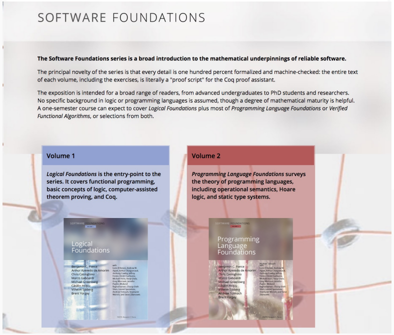
Well-Understood for Functional Languages (more or less)

Mechanized Metatheory for the Masses: The POPLMARK Challenge

Brian E. Aydemir[1], Aaron Bohannon[1], Matthew Fairbairn[2], J. Nathan Foster[1], Benjamin C. Pierce[1], Peter Sewell[2], Dimitrios Vytiniotis[1], Geoffrey Washburn[1], Stephanie Weirich[1], and Steve Zdancewic[1]

[1] Department of Computer and Information Science, University of Pennsylvania
[2] Computer Laboratory, University of Cambridge

Subversion Revision: 171
Document generated on: May 11, 2005 at 15:53



POPLMark Reloaded: Mechanizing Proofs by Logical Relations

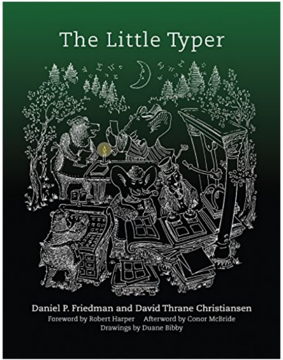
ANDREAS ABEL
Department of Computer Science and Engineering, Gothenburg University, Sweden

GUILLAUME ALLAIS
iCIS, Radboud University, Nijmegen, Netherlands

ALIYA HAMEER and BRIGITTE PIENTKA
School of Computer Science, McGill University, Canada

ALBERTO MOMIGLIANO
Department of Computer Science, University of Milan, Italy

STEVEN SCHÄFER and KATHRIN STARK



Programming Language Foundations in Agda Table of Contents Getting Starte

Table of Contents

This book is an introduction to programming language theory using the proof assistant Agda.

Comments on all matters—organisation, material to add, material to remove, parts that require better explanation, good exercises, errors, and typos—are welcome. The book repository is on GitHub. Pull requests are encouraged.

Front matter

Part 1: Logical Foundations

But Mechanization of Process Calculi Remain Very Difficult

But Mechanization of Process Calculi Remain Very Difficult

1995 *T. Melham:* A mechanized theory of the π-calculus in HOL. Nordic Journal of Computing, 1(1):50–76

1997 *D. Hirschkoff*: A full formalisation of π-calculus theory in the calculus of con- structions. TPHOL'97

2001 *Christine Röckl, Daniel Hirschkoff, Stefan Berghofer:* Higher-Order Abstract Syntax with Induction in Isabelle/HOL: Formalizing the pi-Calculus and Mechanizing the Theory of Contexts, Fossacs'01

The Top 75% Are Tedious ...

> *"approaches in the literature to the implementation of the π-calculus had adopted either a direct first-order encoding or had dropped names tout court in favour of de Brujin indexes. The user is hence overwhelmed by technical details and lemmata about equivalence free names operators, substitution functions and so ... out 600 of 800 proved lemmata concern the technical details of index handling."*

F. Honsell, M. Miculan, I. Scagnetto, Pi-calculus in (Co)Inductive Type Theory, Theoretical Computer Science, Vol 253, 2001

Session Types in a Nutshell

- Message passing calculus (like the π-calculus) with a typing discipline for structured interactions

"Session types are applied to a wide range of problems, and their properties, such as deadlock-freedom, are well studied. These calculi are very expressive, and rather complex, with features like: shared and linear communication channels, name passing, and fresh name generation. Given this complexity, it is not surprising that some innocent looking extensions violated the type safety properties of the calculus in several literature (as pointed out by [23]). "

[Castro-Perez, Ferreira, and Yoshida'20]

Recent Efforts on Mechanizing Session Types

- Functional language with session-typed communication [Thiemann'19] **de Bruijn**
- Type-checking session-typed pi-calculus with Coq [Zalakain'19]
  combination of linearity/wf checks and explicit contexts
- Π with left-overs: A Mechanization in Agda. [Zalakain and Dardha'21]
  explicit contexts with left-over constraints
- General framework of mechanizing session-typed process calculi in Coq. [Castro-Perez, Ferreira, and Yoshida'20] **locally nameless**
- Concurrent Calculi Formalisation Benchmark [Ferreira, et. al. 2023]

It's been difficult to encode session type systems .... The fact that there are different formulations (GV, Multi-Party Session Types, etc.) isn't helping.

A Logical View of Session Types

(Classical) Linear Logic – A Curry Howard View of Session Types

"A logic without weakening or contraction" [Girard 87]

$$\vdash A_1, \ldots, A_n$$

$A, B ::= A \otimes B$	(Tensor; multiplicative conjunction)
$\mid A \mathbin{\&} B$	(With/and; additive conjunction)
$\mid A \mathbin{\rotatebox[origin=c]{180}{\&}} B$	(Par; multiplicative disjunction)
$\mid A \oplus B$	(Plus; additive disjunction)
$\mid 1$	(Unit of $\otimes$)
$\mid \bot$	(Unit of $\rotatebox[origin=c]{180}{\&}$)

Linear negation $A^\perp$

Linear Propositions as Session Types [Caires, Pfenning'10][Wadler'12]

Linear Logic	Message Passing Concurrency

Linear Propositions as Session Types [Caires, Pfenning'10][Wadler'12]

Linear Logic	Message Passing Concurrency
Assumptions	Channels
Linear Propositions	Session types
Sequents	Process Typing
Proofs	Well-Typed Processes
Cut	Parallel Composition
Cut elimination	Communication

Type Judgements - Classical Processes (CP) [Wadler'12]

Identity and Composition

$$\frac{}{\mathsf{fwd}\ x\ y \vdash x : A, y : A^{\perp}}\ (\textsc{Id}) \qquad \frac{P \vdash \Delta_1, x : A \quad Q \vdash \Delta_2, x : A^{\perp}}{\nu x{:}A.(P \parallel Q) \vdash \Delta_1, \Delta_2}\ (\textsc{Cut})$$

- Identity – Forwarding future communication between channels
- Composition – spawns processes P and Q that communicate along a bound private channel x.

Challenge

- Variable binding
- Splitting context and ensuring channel that no other channels are shared

Type Judgements - Classical Processes (CP) [Wadler'12]

Internal and External Choice

Duality of Plus and (Additive) With/And

$$(A \otimes B)^{\perp} \quad = \quad A^{\perp} \parr B^{\perp}$$

$$\frac{P \vdash \Delta, x : A}{\mathsf{inl}^x\ P \vdash \Delta, x : A \oplus B}\ (\oplus_1) \qquad \frac{P \vdash \Delta, x : B}{\mathsf{inr}^x\ P \vdash \Delta, x : A \oplus B}\ (\oplus_2)$$

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

- Send a "left" or "right" choice over x and continue with P
- Based on the received choice choose process P or Q

Challenges

- "Update" the type associated with the channel

Type Judgements - Classical Processes (CP) [Wadler'12]

Channel Transmission

Duality of (Multiplicative) Par and Tensor

$$(A \parr B)^\perp \quad = \quad A^\perp \otimes B^\perp$$

$$\frac{P \vdash \Delta_1, y : A \quad Q \vdash \Delta_2, x : B}{\mathsf{out}\; x\, y; (P \parallel Q) \vdash \Delta_1, \Delta_2, x : A \otimes B}\ (\otimes) \qquad \frac{P \vdash \Delta, x : B, y : A}{\mathsf{inp}\; x\, y; P \vdash \Delta, x : A \parr B}\ (\parr)$$

- **Out** : Sends a channel name y across the channel x, and spawns concurrent processes P and Q that provide channel y and x
- **In** : Receives a channel over x, binds it to a fresh name y, and proceeds as P.

Challenges

- "Update" the type associated with the channel x
- Variable binding
- Splitting context to ensure channels are not shared between P and Q

Type Judgements - Classical Processes (CP) [Wadler'12]

Termination

$$\frac{}{\text{close } x \vdash x : 1}\ (1) \qquad \frac{P \vdash \Delta}{\text{wait } x; P \vdash \Delta, x : \bot}\ (\bot)$$

- Termination and Waiting for termination

Challenges

- Ensure that there are no other left-over channels
- Consume channel x

Reductions and Commuting Conversions

Reduction and Principal Reduction:

$$\frac{}{\nu x{:}A.(\mathsf{fwd}\ x\ y \parallel Q) \Rightarrow_{CP} [y/x]Q}\ (\beta_{\text{FWD}})$$

$$\frac{}{\nu x{:}A \oplus B.(\mathsf{inl}^x\ P' \parallel \mathsf{case}\ x\ (Q_1,\ Q_2)) \Rightarrow_{CP} \nu x{:}A.(P' \parallel Q_1)}\ (\beta_{\text{INL}})$$

Commuting Conversion:

$$\frac{}{\nu z{:}C.(\mathsf{inl}^x\ P' \parallel Q) \Rightarrow_{CP} \mathsf{inl}^x\ \nu z{:}C.(P' \parallel Q)}\ (\kappa_{\text{INL}})$$

Congruence Rules including Associativity and Communicativity of parallel process composition

Main Property

Theorem: Type Preservation

If $P \vdash \Delta$ and $P \Rightarrow_{CP} Q$, then $Q \vdash \Delta$.

Two main challenges

- How to deal with contexts?
- How to deal with channel names?

They are somewhat related ...

Goal: Leverage HOAS

- Leverage binding infrastructure for modelling channel dependencies in processes
  (inherit α-renaming, substitution operation, easy checking for variable dependencies, etc.)
- Leverage (ambient) context and hypothetical judgments
  (inherit substitution property, context management including weakening, contraction, uniqueness)

Problem

We have a linear context and we re-use channel names

Goal: Leverage HOAS

- Leverage binding infrastructure for modelling channel dependencies in processes
  (inherit α-renaming, substitution operation, easy checking for variable dependencies, etc.)

- Leverage (ambient) context and hypothetical judgments
  (inherit substitution property, context management including weakening, contraction, uniqueness)

Solution

1. **Linear Context:**
   - *Adapt idea from Crary [ICFP'12] to session types*
   - *Define and use linearity predicate to ensure a channel name occurs only once*
   - *If a channel name occurs once, any assumption involving it can only be used once*
2. **Continuation Channels to avoid re-using channel names**

Listen to LF

"You have to listen to the logical framework, as it were, and take its advice in guiding you towards a better way to formulate your system. We learned this lesson many years ago when we first invented LF — the exercise of formalizing a logic in LF does wonders for the logic."

Bob Harper's post to the POPLmark-list 2 May 2006

A Structural Process Calculus (SCP)

[Sano, Kavanagh, Pientka'23]

Making the continuation channel explicit in SCP

Previous rule for $A \oplus B$ (Plus; additive disjunction)

$$\frac{P \vdash \Delta, x : A}{\mathsf{inl}^x\ P \vdash \Delta, x : A \oplus B}\ (\oplus_1) \qquad \frac{P \vdash \Delta, x : B}{\mathsf{inr}^x\ P \vdash \Delta, x : A \oplus B}\ (\oplus_2)$$

Restating $\mathsf{inl}^x\ P$ **and** $\mathsf{inr}^x\ P$ **with an explicit continuation channel** w

$$\frac{\mathtt{P} \Vdash \Gamma, x : A \oplus B, w : A}{\mathtt{inl}\ x;\ w.\mathtt{P} \Vdash \Gamma, x : A \oplus B}\ [\oplus_1] \qquad \frac{\mathtt{P} \Vdash \Gamma, x : A \oplus B, w : B}{\mathtt{inr}\ x;\ w.\mathtt{P} \Vdash \Gamma, x : A \oplus B}\ [\oplus_2]$$

Making the continuation channel explicit in SCP

Previous rule for $A \oplus B$ (Plus; additive disjunction)

$$\frac{P \vdash \Delta, x : A}{\mathsf{inl}^x\ P \vdash \Delta, x : A \oplus B}\ (\oplus_1) \qquad \frac{P \vdash \Delta, x : B}{\mathsf{inr}^x\ P \vdash \Delta, x : A \oplus B}\ (\oplus_2)$$

Restating $\mathsf{inl}^x\ P$ and $\mathsf{inr}^x\ P$ with an explicit continuation channel w

$$\frac{\mathtt{P} \Vdash \Gamma, x : A \oplus B, w : A}{\mathtt{inl}\ x;\ w.\mathtt{P} \Vdash \Gamma, x : A \oplus B}\ [\oplus_1] \qquad \frac{\mathtt{P} \Vdash \Gamma, x : A \oplus B, w : B}{\mathtt{inr}\ x;\ w.\mathtt{P} \Vdash \Gamma, x : A \oplus B}\ [\oplus_2]$$

Using explicit continuation channel also for $\mathsf{case}\ x\ (P,\ Q)$

$$\frac{\mathtt{P} \Vdash \Gamma, x : A \,\&\, B, w : A \qquad \mathtt{Q} \Vdash \Gamma, x : A \,\&\, B, w : B}{\mathtt{case}\ x\ (w.\mathtt{P},\ w.\mathtt{Q}) \Vdash \Gamma, x : A \,\&\, B}\ [\&]$$

Ensuring internal bindings are linear in SCP

Idea: Check linearity for fresh channels

$$\frac{\texttt{P} \Vdash \Gamma, x : A \quad \mathsf{lin}(x,\ \texttt{P}) \quad \texttt{Q} \Vdash \Gamma, x : A^{\perp} \quad \mathsf{lin}(x,\ \texttt{Q})}{\nu x{:}A.(\texttt{P} \parallel \texttt{Q}) \Vdash \Gamma}\ [\texttt{Cut}]$$

Ensuring internal bindings are linear in SCP

Idea: Check linearity for fresh channels

$$\frac{\mathtt{P} \Vdash \Gamma, x : A \quad \mathsf{lin}(x,\ \mathtt{P}) \quad \mathtt{Q} \Vdash \Gamma, x : A^{\perp} \quad \mathsf{lin}(x,\ \mathtt{Q})}{\nu x{:}A.(\mathtt{P} \parallel \mathtt{Q}) \Vdash \Gamma}\ [\mathtt{Cut}]$$

How does this work for when we reach a channel name?

$$\frac{}{\mathsf{close}\ x \Vdash \Gamma, x : 1}\ [\mathtt{1}] \qquad \frac{\mathtt{P} \Vdash \Gamma, x : \perp}{\mathsf{wait}\ x; \mathtt{P} \Vdash \Gamma, x : \perp}\ [\perp]$$

Localize Linearity - Linearity Predicate

fn($\texttt{P}$) - "The set of free channel names in $\texttt{P}$."

lin(x, $\texttt{P}$) - "Channel x and its continuations are used linearly in $\texttt{P}$."

Localize Linearity - Linearity Predicate

fn($\texttt{P}$) - "The set of free channel names in $\texttt{P}$."

lin(x, $\texttt{P}$) - "Channel x and its continuations are used linearly in $\texttt{P}$."

$$\frac{}{\mathsf{lin}(x,\ \texttt{close}\ x)}\ L_{\texttt{close}} \qquad \frac{\mathsf{lin}(w,\ \texttt{P}) \quad x \notin \mathsf{fn}(\texttt{P})}{\mathsf{lin}(x,\ \texttt{inl}\ x;\ w.\texttt{P})}\ L_{\texttt{inl}}$$

Localize Linearity - Linearity Predicate

$\mathrm{fn}(\mathtt{P})$ - "The set of free channel names in $\mathtt{P}$."

$\mathrm{lin}(x,\ \mathtt{P})$ - "Channel x and its continuations are used linearly in $\mathtt{P}$."

$$\frac{}{\mathrm{lin}(x,\ \mathtt{close}\ x)}\ L_{\mathtt{close}} \qquad \frac{\mathrm{lin}(w,\ \mathtt{P}) \quad x \notin \mathrm{fn}(\mathtt{P})}{\mathrm{lin}(x,\ \mathtt{inl}\ x;\ w.\mathtt{P})}\ L_{\mathtt{inl}}$$

$$\frac{\mathrm{lin}(z,\ \mathtt{P}) \quad z \notin \mathrm{fn}(\mathtt{Q})}{\mathrm{lin}(z,\ \nu x{:}A.(\mathtt{P} \parallel \mathtt{Q}))}\ L_{\nu 1} \qquad \frac{\mathrm{lin}(z,\ \mathtt{Q}) \quad z \notin \mathrm{fn}(\mathtt{P})}{\mathrm{lin}(z,\ \nu x{:}A.(\mathtt{P} \parallel \mathtt{Q}))}\ L_{\nu 2}$$

Localize Linearity - Linearity Predicate

$\mathrm{fn}(\mathtt{P})$ - "The set of free channel names in $\mathtt{P}$."

$\mathrm{lin}(x,\ \mathtt{P})$ - "Channel x and its continuations are used linearly in $\mathtt{P}$."

$$\frac{}{\mathrm{lin}(x,\ \mathtt{close}\ x)}\ L_{\mathtt{close}} \qquad \frac{\mathrm{lin}(w,\ \mathtt{P}) \quad x \notin \mathrm{fn}(\mathtt{P})}{\mathrm{lin}(x,\ \mathtt{inl}\ x;\ w.\mathtt{P})}\ L_{\mathtt{inl}}$$

$$\frac{\mathrm{lin}(z,\ \mathtt{P}) \quad z \notin \mathrm{fn}(\mathtt{Q})}{\mathrm{lin}(z,\ \nu x{:}A.(\mathtt{P} \parallel \mathtt{Q}))}\ L_{\nu 1} \qquad \frac{\mathrm{lin}(z,\ \mathtt{Q}) \quad z \notin \mathrm{fn}(\mathtt{P})}{\mathrm{lin}(z,\ \nu x{:}A.(\mathtt{P} \parallel \mathtt{Q}))}\ L_{\nu 2}$$

Note: We can check $x \notin \mathrm{fn}(\mathtt{P})$ exploiting the powers of higher-order unification in HOAS systems where we check and encode variable dependencies.

Key Principle to Allow for Extensions

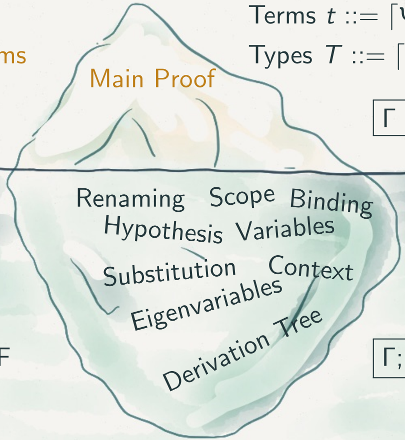
- **Does the construct bind any new linear channels?**
  If yes, then the typing judgment must check their linearity.
- **Does the construct requires the absence of other linear assumptions?**
  If yes, then there should be no congruence rules for the linearity predicate.
- **Does the construct use a continuation channel?**
  If yes, then the linearity predicate should check that the continuation channel is used linearly. Otherwise,the linearity predicate should be an axiom.
- **Are linear channels shared between subterms composed by the construct?**
  If they are not shared, then the linearity predicate must ensure that no sharing occurs.

Encoding SCP in Beluga

The Tip of the Iceberg: Beluga [POPL'08, IJCAR'10, POPL'12, CADE'15, ICFP'16,...]

PPDP Test of Time Award in 2018

Proofs
as Functional Programs



Terms $t ::= \lceil \Psi \vdash M \rceil \mid \ldots$

Types $T ::= \lceil \Psi \vdash A \rceil \mid \ldots$

$\boxed{\Gamma \Vdash t : T}$

Contextual
Logical Framework LF

$\boxed{\Gamma; \Psi \Vdash M : A}$

Encoding Processes using (weak) HOAS

$$\text{Processes} \quad P, Q \quad ::= \quad \texttt{fwd}\ x\ y \mid \mathsf{close}\ x \mid \mathsf{wait}\ x; \texttt{P} \mid \nu x{:}A.(\texttt{P} \parallel \texttt{Q}) \mid$$
$$\texttt{inl}\ x;\ \ w.\texttt{P} \mid \texttt{inr}\ x;\ \ w.\texttt{P} \mid \texttt{case}\ x\ (w.\texttt{P},\ \ w.\texttt{Q}) \mid$$
$$\texttt{out}\ x;\ (y.\texttt{P} \parallel w.\texttt{Q}) \mid \texttt{inp}\ x\ (w.y.\texttt{P})$$

LF Representation

fwd : name → name → proc.
close : name → proc.
wait : name → proc → proc.
pcomp : tp → (name → proc) → (name → proc) → proc.
inl : name → (name → proc) → proc.
inr : name → (name → proc) → proc.
case : name → (name → proc) → (name → proc) → proc.
inp : name → (name → name → proc) → proc.
out : name → (name → proc) → (name → proc) → proc.

Representing Session Typing in LF

wtp: **proc** $\to$ **type**

$$P \vdash x_1 : A_1, \ldots, x_n : A_n$$

```
wtp_fwd : dual A A'
    → {X:name}hyp X A
    → {Y:name}hyp Y A'
    → wtp (fwd X Y).
```

$$\frac{}{\texttt{fwd}\ x\ y \Vdash \Gamma, x : A, y : A^{\perp}}\ [\texttt{Id}]$$

```
wtp_close : {X:name}hyp X 1
  → wtp (close X).
```

$$\frac{}{\mathsf{close}\ x \Vdash \Gamma, x : 1}\ [\texttt{1}]$$

```
wtp_pcomp : dual A A'
  → ({x:name} hyp x A → wtp (P x))
  → ({x:name} hyp x A' → wtp (Q x))
  → linear P → linear Q
  → wtp (pcomp A P Q).
```

$$\frac{\texttt{P} \Vdash \Gamma, x : A \quad \mathsf{lin}(x,\ \texttt{P}) \quad \texttt{Q} \Vdash \Gamma, x : A^{\perp} \quad \mathsf{lin}(x,\ \texttt{Q})}{\nu x{:}A.(\texttt{P} \parallel \texttt{Q}) \Vdash \Gamma}\ [\texttt{Cut}]$$

Encoding the Linearity Predicate

linear : (**name** → **proc**) → **type**.

$$\mathrm{lin}(x,\ P)$$

l_close : **linear** (λx. **close** x).

$$\frac{}{\mathrm{lin}(x,\ \texttt{close}\ x)}\ L_{\texttt{close}}$$

l_wait : **linear** (λx. **wait** x P).

$$\frac{x \notin \mathrm{fn}(\mathtt{P})}{\mathrm{lin}(x,\ \texttt{wait}\ x;\ \mathtt{P})}\ L_{\texttt{wait}}$$

l_inl : **linear** P
→ **linear** (λx. **inl** x P).

$$\frac{\mathrm{lin}(w,\ \mathtt{P}) \quad x \notin \mathrm{fn}(\mathtt{P})}{\mathrm{lin}(x,\ \texttt{inl}\ x;\ w.\mathtt{P})}\ L_{\texttt{inl}}$$

l_out : **linear** Q
→ **linear** (λx. **out** x P Q).

$$\frac{\mathrm{lin}(w,\ \mathtt{Q}) \quad x \notin \mathrm{fn}(\mathtt{P}) \cup \mathrm{fn}(\mathtt{Q})}{\mathrm{lin}(x,\ \texttt{out}\ x;\ (y.\mathtt{P} \parallel w.\mathtt{Q}))}\ L_{\texttt{out}}$$

Encoding Reductions

For example:

$$\frac{}{\nu x{:}A \oplus B.(\texttt{inl}\ x;\ \ w.\texttt{P} \parallel \texttt{case}\ x\ (w.\texttt{Q},\ \ w.\texttt{R})) \Rightarrow_{SCP} \nu w{:}A.(\texttt{P} \parallel \texttt{Q})}\ [\beta_{\texttt{inl}}]$$

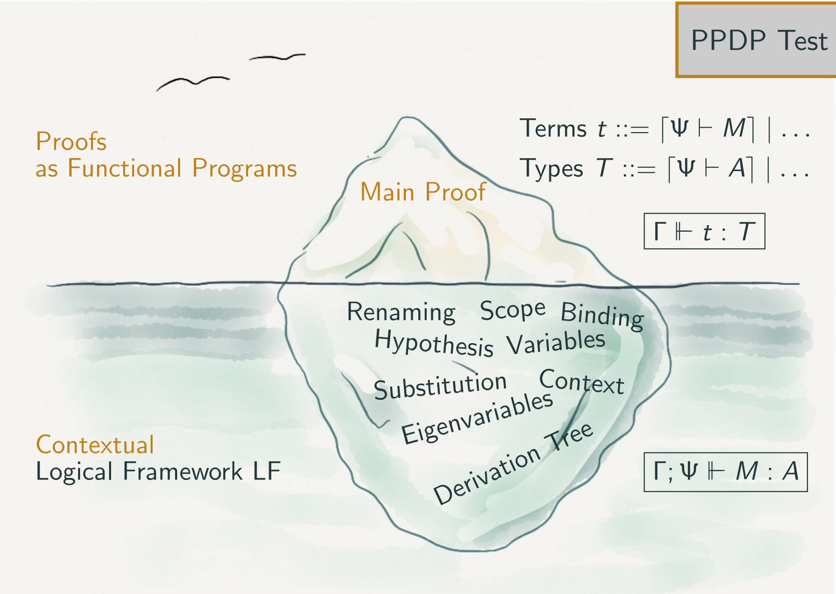
is translated to:

β**inl** : **step** (**pcomp** (A $\oplus$B) (λx. **inl** x P) (λx. **case** x Q R)) (**pcomp** A P Q).

Implementing Type Preservation

The Tip of the Iceberg: Beluga [POPL'08, IJCAR'10, POPL'12, CADE'15, ICFP'16,...]

PPDP Test of Time Award in 2018



Main Property – Revisited in Beluga

Theorem: Type Preservation of CP

If $P \vdash \Delta$ and $P \Rightarrow_{CP} Q$, then $Q \vdash \Delta$.

```
rec wtp_s : (Γ : ctx) [Γ ⊢ wtp P ] → [Γ ⊢ step P Q ] → [Γ ⊢ wtp Q ]
```

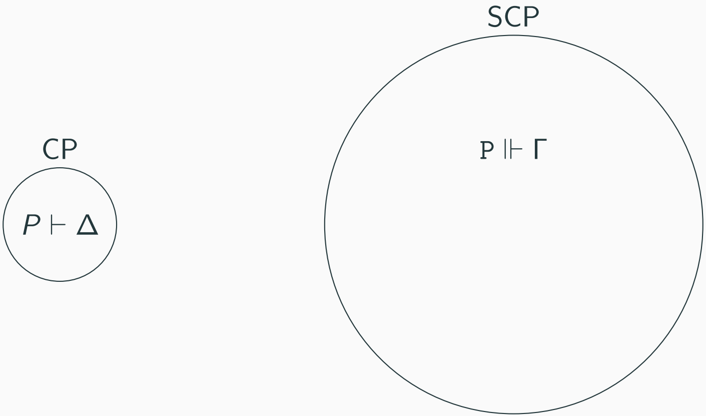
and

```
rec lin_s : (Γ : ctx) [Γ, x:name, h:hyp x A[] ⊢ wtp P[..,x] ]
                    → [Γ ⊢ linear (λx. P [.., x ])]
                    → [Γ, x:name ⊢ step P[..,x] Q [.., x] ]
                    → [Γ ⊢ linear (λx. Q [.., x ])]
```

- Uses contextual objects
- Abstract over the set of channel names and assumptions about channels

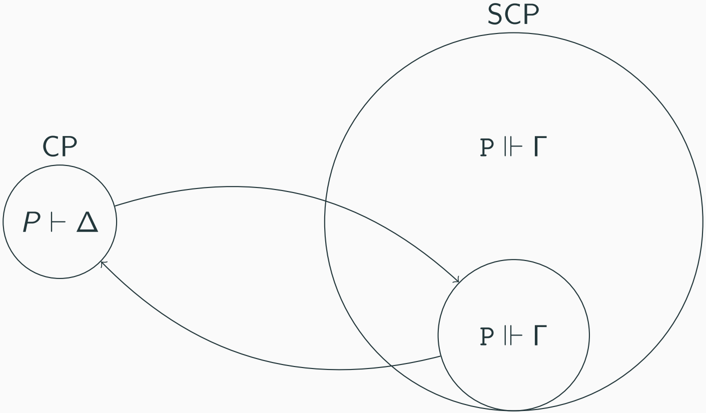
Summary of SCP

- Unrestricted context and linearity predicate precisely characterize linearity.
- Continuation channels make channel dependencies explicit.
- Equivalence between CP and SCP



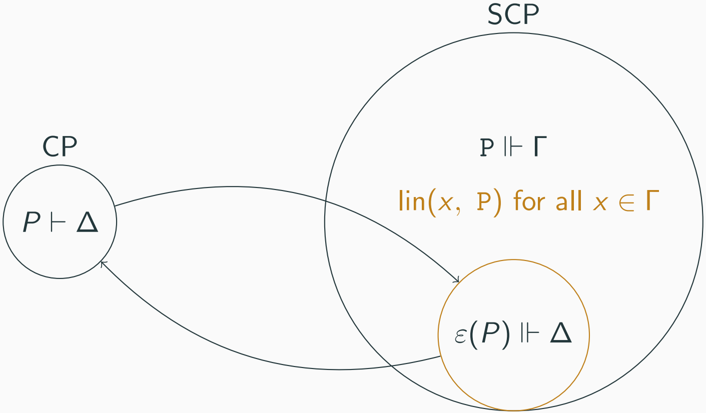
Summary of SCP

- Unrestricted context and linearity predicate precisely characterize linearity.
- Continuation channels make channel dependencies explicit.
- Equivalence between CP and SCP



Summary of SCP

- Unrestricted context and linearity predicate precisely characterize linearity.
- Continuation channels make channel dependencies explicit.
- Equivalence between CP and SCP



Mechanizations of Sub-Structural Systems in Beluga

- SCP [C. Sano, R. Kavanagh] **Type Preservation**

  About 10 lemmas, in particular strengthening lemmas that would be difficult to state in a system like Twelf; (1+1)x2 main theorems for reductions and equivalences

- CP with explicit contexts and continuation channels [D. Zackon]

  approx. 90 lemmas just about context management in CP **Equivalence to SCP**

  Type Preservation

- CP with explicit contexts and updating of channel names [D. Zackon]

  Equivalence of between two formulations of CP

What's next?

What's next?

ToDo 1: Exploring how reusable the infrastructure for explicitly managing contexts is in other sub-structural settings

ToDo 2: Mechanize deadlock freedom for SCP (and CP) (Seems relatively straightforward)

ToDo 3: Add more type constructors such as exponentials $!A$ and $?A$ to the mechanization

ToDo 4: Mechanize other theorems about session types

ToDo 5: Mechanize other session type systems [Vasconcelos'12] and Concurrent Calculi Formalisation Benchmark [Ferreira, et. al. 2023]

What I Learned

Lesson 1: Explicit linearity predicate is surprisingly effective.

Lesson 2: HOAS is a surprisingly effective way to re-think and encode sub-structural systems (channel name binding, context management, free variable condition checks).

Lesson 3: Apply these ideas to other sub-structural systems such as for example Quiper (Quantum Programming Language)

Lesson 4: The approach should also scale to systems such as Coq where we can take advantage of existing infrastructure for contexts and variable bindings for intuitionistic systems (i.e. re-use the Auto-Subst library; still need to implement free variable checks)

Lesson 5: Most session types are simply typed – can we provide stronger static guarantees about processes? (ongoing work with Ryan Kavanagh)