Typing a linear π-calculus

Uma Zalakain
University of Glasgow, Scotland
u.zalakain.1@research.gla.ac.uk

Ornela Dardha
University of Glasgow, Scotland
ornela.dardha@glasgow.ac.uk

remove todo notes

Abstract

We present the syntax, operational semantics, and typing rules of a π-calculus with linear and shared types. We use leftover typing [1] to encode our typing rules in a way that propagates linearity constraints into process continuations. We generalize the algebras on multiplicities using indexed sets of *partial commutative monoids*, allowing the user to choose a mix of linear, affine, gradual and shared typing. We provide framing, weakening and strengthening proofs that we then use to prove subject congruence. We show that the type system is stable under substitution and prove subject reduction.

This formalization has been fully mechanized with Agda and is available at `https://github.com/umazalakain/typing-linear-pi`.

2012 ACM Subject Classification Theory of computation → Process calculi

Keywords and phrases pi calculus, linear, types, concurrency

Digital Object Identifier 10.4230/LIPIcs...

Supplement Material `https://github/umazalakain/typing-linear-pi`

Acknowledgements I want to thank . . .

1 Introduction

The π-calculusmodels communication.

why resource-aware typing

extensional typing rules for a given syntax and operational semantics

leftover typing

1.1 Contribution

Machine verified formalisation of the linear pi calculus
Typing with leftovers applied to the pi calculus
Abstraction over multiplicities
Full formalisation available in Agda

1.2 Notation

$$\dfrac{}{\mathbb{N} : Set} \qquad \dfrac{}{0 : \mathbb{N}} \qquad \dfrac{n : \mathbb{N}}{1{+}n : \mathbb{N}}$$

Figure 1 Notation used in this paper

Double rule for type-level definitions. We omit universe levels for brevity. Constructors are `teletyped` unless they are symbols. Some constructors are shared across types. They can however always be disambiguated through the type of the goal.

2 Related work

[?] polymorphic tokens, HOAS

[?]

3 Syntax

variable references (strings, locally named, de Bruijn)

allows to ignore alpha conversion, or proofs of inequality between strings

strings to maybe de Bruijn, names can be kept in context as well, just not doing it

$$\frac{n : \mathbb{N}}{\mathrm{VAR}_n : \mathit{Set}} \qquad \frac{n : \mathbb{N}}{0 : \mathrm{VAR}_n} \qquad \frac{x : \mathrm{VAR}_n}{{}_{1+}x : \mathrm{VAR}_{1+n}}$$

Figure 2 Types of size n

$$\frac{n : \mathbb{N}}{\mathrm{PROCESS}_n : \mathit{Set}}$$

$$\begin{aligned} \mathrm{PROCESS}_n ::= {} & \mathbf{0}_n \\ & \mid \boldsymbol{\nu}\, \mathrm{PROCESS}_{1+n} \\ & \mid \mathrm{PROCESS}_n \parallel \mathrm{PROCESS}_n \\ & \mid \mathrm{VAR}_n\, ()\, \mathrm{PROCESS}_{1+n} \\ & \mid \mathrm{VAR}_n\, \langle \mathrm{VAR}_n \rangle\, \mathrm{PROCESS}_n \end{aligned}$$

Figure 3 Well-scoped grammar using de Bruijn indices

4 Semantics

4.1 Structural congruence

congruence relationship indexed by recursive tree

4.2 Reduction

keeping track of the variable on which communication occurs

$$\frac{}{P \equiv_n Q : Set} \qquad \frac{}{\texttt{comp}-\texttt{assoc} : P \parallel Q \parallel R \equiv_n P \parallel Q \parallel R}$$

$$\frac{}{\texttt{comp}-\texttt{sym} : P \parallel Q \equiv_n Q \parallel P} \qquad \frac{}{\texttt{comp}-\texttt{end} : P \parallel \mathbf{0}_n \equiv_n P}$$

$$\frac{}{\texttt{scope}-\texttt{end} : \boldsymbol{\nu}\, \mathbf{0}_{1+n} \equiv_n \mathbf{0}_n} \qquad \frac{P\ Q : \mathrm{PROCESS}_{1+n} \qquad uQ : \mathrm{UNUSED}_0\, Q}{\texttt{scope}-\texttt{ext} : \boldsymbol{\nu}\,(P \parallel Q) \equiv_n (\boldsymbol{\nu}\, P) \parallel lower\ 0\ \ Q\ uQ}$$

$$\frac{P : \mathrm{PROCESS}_{1+1+n}}{\texttt{scope}-\texttt{comm} : \boldsymbol{\nu}\,\boldsymbol{\nu}\, P \equiv_n \boldsymbol{\nu}\,\boldsymbol{\nu}\, swap\ 0\ P}$$

Figure 4 Structural rewriting rules indexed by recursion tree. $P\ Q\ R : \mathrm{PROCESS}_n$ where P, Q, or R have been omitted for brevity.

5 Linear typing rules

5.1 Multiplicities

A type system with both linear and shared resources has multiplicities 0, 1 and ω.

alignment, get rid of $1\cdot$, $\cdot - join$, $\cdot - compute^l$

5.1.1 Example type systems

Shared. Gradual. Affine. Linear.

5.2 Variable references

5.3 Contexts

two-layered approach: types on one hand, capabilities on the other removing from context vs keeping in context but marking it used

5.4 Typing with leftovers

5.4.1 Typing relation

Variable references as proofs of capability

Context splits at each variable reference

6 Subject reduction

6.1 Framing

6.2 Weakening

Order preserving embeddings model a series of insertions. We only ever need one insertion to prove subject congruence, but there is no loss of generality.

$$\frac{}{\mathrm{REC} : \mathit{Set}} \qquad \frac{}{\texttt{zero} : \mathrm{REC}} \qquad \frac{r : \mathrm{REC}}{\texttt{one}\ r : \mathrm{REC}} \qquad \frac{r\ s : \mathrm{REC}}{\texttt{two}\ r\ s : \mathrm{REC}}$$

$$\frac{P\,Q : \mathrm{PROCESS}_n \qquad r : \mathrm{REC}}{P =_r Q : \mathit{Set}} \qquad \frac{P \equiv_n Q}{\texttt{struct} : P =_{\texttt{zero}} Q}$$

$$\frac{P\ P' : \mathrm{PROCESS}_{1+n} \qquad P =_r P'}{\texttt{cong} - \texttt{scope} : \boldsymbol{\nu}\, P =_{\texttt{one}\ r} \boldsymbol{\nu}\, P'} \qquad \frac{P =_r P'}{\texttt{cong} - \texttt{comp} : P \parallel Q =_{\texttt{one}\ r} P' \parallel Q}$$

$$\frac{P\ P' : \mathrm{PROCESS}_{1+n} \qquad P =_r P'}{\texttt{cong} - \texttt{recv} : x\,()\,P =_{\texttt{one}\ r} x\,()\,P'} \qquad \frac{P =_r P'}{\texttt{cong} - \texttt{send} : x\,\langle y \rangle\,P =_{\texttt{one}\ r} x\,\langle y \rangle\,P'}$$

$$\frac{P : \mathrm{PROCESS}_n}{\texttt{refl} : P =_{\texttt{zero}} P} \qquad \frac{P =_r Q}{\texttt{sym} : Q =_{\texttt{one}\ r} P} \qquad \frac{P =_r Q \qquad Q =_s R}{\texttt{trans} : P =_{\texttt{two}\ r\ s} R}$$

Figure 5 Structural rewriting rules lifted to a congruent equivalence relation. $P\ P'\ Q\ R : \mathrm{PROCESS}_n$ where P, P', Q, or R have been omitted for brevity.

6.3 Strengthening

6.4 Swapping

6.5 Substitution

7 Future work

Work that will be done time permiting:

Affine types
Proof of progress
Product types
Sum types
Decidable typechecking
Soundness and completeness with respect to an alternative formalization.
Encoding of session types

References

1 Guillaume Allais. Typing with Leftovers - A mechanization of Intuitionistic Multiplicative-Additive Linear Logic. page 22 pages, 2018. http://drops.dagstuhl.de/opus/volltexte/2018/10049/. doi:10.4230/lipics.types.2017.1.

$$\frac{n : \mathbb{N}}{\text{CHANNEL}_n : \mathit{Set}} \qquad \frac{}{\texttt{nothing} : \text{CHANNEL}_n} \qquad \frac{i : \text{VAR}_n}{\texttt{just}\ i : \text{CHANNEL}_n}$$

$$\frac{i : \text{CHANNEL}_n \qquad P\ Q : \text{PROCESS}_n}{P \longrightarrow_i Q : \mathit{Set}}$$

$$\frac{i\ j : \text{VAR}_n \qquad P : \text{PROCESS}_{1+n} \qquad Q : \text{PROCESS}_n \qquad uP : \text{UNUSED}_0\ P}{\texttt{comm} : i\,()\,P \parallel i\,\langle j\rangle\,Q \longrightarrow_{just\ i} \mathit{lower}\ 0\ P[j/0]\ uP \parallel Q}$$

$$\frac{P \longrightarrow_i P'}{\texttt{par} : P \parallel Q \longrightarrow_i P' \parallel Q} \qquad \frac{P \longrightarrow_i Q}{\texttt{res} : \boldsymbol{\nu}\, P \longrightarrow_{dec\ i} \boldsymbol{\nu}\, Q} \qquad \frac{P = P' \qquad P' \longrightarrow_i Q}{\texttt{struct} : P \longrightarrow_i Q}$$

Figure 6 Operational semantics indexed by reducing channel

$$\begin{aligned}
\mathtt{0}\cdot &: C \\
+\cdot &: C \\
-\cdot &: C \\
_ := _ \cdot _ &: C \to C \to C \to \mathit{Set} \\
\mathtt{1}\cdot &: C \\
\cdot - \texttt{join} &: \forall\{xyz\} \to x := y \cdot +\cdot \to x := z \cdot -\cdot \to \exists w.(x := w \cdot 1\cdot) \\
\cdot - \texttt{compute} &: \forall yz \to \mathit{Dec}(\exists x.(x := y \cdot z)) \\
\cdot - \texttt{compute}^{\mathtt{l}} &: \forall xz \to \mathit{Dec}(\exists y.(x := y \cdot z)) \\
\cdot - \texttt{unique} &: \forall\{xx'yz\} \to x' := y \cdot z \to x := y \cdot z \to x' \equiv x \\
\cdot - \texttt{unique}^{\mathtt{l}} &: \forall\{xyy'z\} \to x := y' \cdot z \to x := y \cdot z \to y' \equiv y \\
\cdot - \texttt{id}^{\mathtt{l}} &: \forall x \to x := 0 \cdot \cdot x \\
\cdot - \texttt{comm} &: \forall\{xyz\} \to x := y \cdot z \to x := z \cdot y \\
\cdot - \texttt{assoc} &: \forall\{xyzuv\} \to x := y \cdot z \to y := u \cdot v \to \exists w.(x := u \cdot w \times w := v \cdot z)
\end{aligned} \tag{1}$$

Figure 7 Partial commutative monoid

$$\begin{aligned}
\text{IDX} &: \mathit{Set} \\
\exists\text{IDX} &: \mathit{Idx} \\
\text{CARRIER} &: \text{IDX} \to \mathit{Set} \\
\text{QUANTIFIERS} &: \forall i : \text{IDX} \to \text{QUANTIFIER}_{\text{CARRIER}_i}
\end{aligned} \tag{2}$$

Figure 8 Indexed set of partial commutative monoids

$$\frac{n : \mathbb{N}}{\text{PRECTX}_n : \mathit{Set}} \qquad \frac{}{[] : \text{PRECTX}_0} \qquad \frac{\gamma : \text{PRECTX}_n \qquad t : \text{TYPE}}{\gamma, t : \text{PRECTX}_{1+n}}$$

$$\frac{n : \mathbb{N}}{\mathrm{IDXS}_n : \mathit{Set}} \qquad \frac{}{[] : \mathrm{IDXS}_0} \qquad \frac{\mathit{is} : \mathrm{IDXS}_n \qquad i : \mathrm{IDX}}{\mathit{is}, i : \mathrm{IDXS}_{1+n}}$$

$$\frac{\mathit{is} : \mathrm{IDXS}_n}{\mathrm{CTX}_{\mathit{is}} : \mathit{Set}} \qquad \frac{}{[] : \mathit{Ctx}_{[]}} \qquad \frac{\Gamma : \mathrm{CTX}_{\mathit{is}} \qquad x : \mathrm{CARRIER}_i}{\Gamma, x : \mathrm{CTX}_{\mathit{is},i}}$$

$$\frac{\gamma : \mathrm{PRECTX}_n \qquad \begin{array}{c}\mathit{is} : \mathrm{IDXS}_n \\ \Gamma : \mathrm{CTX}_{\mathit{is}}\end{array} \qquad t : \mathrm{TYPE} \qquad x : \mathrm{CARRIER}_i \qquad \Delta : \mathrm{CTX}_{\mathit{is}}}{\gamma \propto \Gamma \ni t \propto x \boxtimes \Delta : \mathit{Set}}$$

$$\frac{\Gamma : \mathrm{CTX}_{\mathit{is}} \qquad y\ z : \mathrm{CARRIER}_i \qquad \mathit{True}(\cdot - \texttt{compute}\ y\ z)}{\texttt{zero} : \gamma, t \propto \Gamma, x \ni t \propto y \boxtimes \Gamma, z}$$

$$\frac{\begin{array}{c}\Gamma : \mathrm{CTX}_{\mathit{is}} \qquad x : \mathrm{CARRIER}_i \qquad x' : \mathrm{CARRIER}_j \qquad \Delta : \mathrm{CTX}_{\mathit{is}} \\ \mathit{loc}_x : \gamma \propto \Gamma \ni t \propto x \boxtimes \Delta\end{array}}{\texttt{suc} : \gamma, t \propto \Gamma, x' \ni t \propto x \boxtimes \Delta, x'}$$

$$\frac{\gamma : \mathrm{PRECTX}_n \qquad \begin{array}{c}\mathit{is} : \mathrm{IDXS}_n \\ \Gamma : \mathrm{CTX}_{\mathit{is}}\end{array} \qquad P : \mathrm{PROCESS}_n \qquad \Delta : \mathrm{CTX}_{\mathit{is}}}{\gamma \propto \Gamma \vdash P \boxtimes \Delta : \mathit{Set}}$$

$$\frac{}{\texttt{end} : \gamma \propto \Gamma \vdash \mathbf{0} \boxtimes \Gamma} \qquad \frac{\begin{array}{l}t : \mathrm{TYPE} \qquad x : \mathrm{CARRIER}_i \qquad y : \mathrm{CARRIER}_j \\ \mathit{cont} : \gamma, C[t \propto x] \propto \Gamma, y \vdash P \boxtimes \Delta, 0.\end{array}}{\texttt{chan} : \gamma \propto \Gamma \vdash \boldsymbol{\nu}\, P \boxtimes \Delta}$$

$$\frac{\begin{array}{l}\mathit{chan}_x : \gamma \propto \Gamma \ni C[t \propto x] \propto + \cdot \boxtimes \Xi \\ \mathit{cont} : \gamma, t \propto \Xi, x \vdash P \boxtimes \Theta, 0 \cdot\end{array}}{\texttt{recv} : \gamma \propto \Gamma \vdash \mathit{toFin}\ \mathit{chan}_x\ ()\ P \boxtimes \Theta} \qquad \frac{\begin{array}{l}\mathit{chan}_x : \gamma \propto \Gamma \ni C[t \propto x] \propto - \cdot \boxtimes \Delta \\ \mathit{loc}_y : \gamma \propto \Delta \ni t \propto x \boxtimes \Xi \\ \mathit{cont} : \gamma \propto \Xi \vdash P \boxtimes \Theta\end{array}}{\texttt{send} : \gamma \propto \Gamma \vdash \mathit{toFin}\ \mathit{chan}_x\ \langle \mathit{toFin}\ \mathit{loc}_y \rangle\ P \boxtimes \Theta}$$

$$\frac{\begin{array}{l}\mathit{left} : \gamma \propto \Gamma \vdash P \boxtimes \Delta \\ \mathit{right} : \gamma \propto \Delta \vdash Q \boxtimes \Xi\end{array}}{\texttt{comp} : \gamma \propto \Gamma \vdash P \parallel Q \boxtimes \Xi}$$