

Barbed Similarity for the π -Calculus in Beluga:

A Case Study in Coinductive Reasoning

Joint work with

Lea Trogni · Università degli Studi di Milano, Dipartimento di Matematica

Alberto Momigliano · Università degli Studi di Milano, Dipartimento di Informatica

Published at [LFMTP 2026](#)

A Beluga formalization of the context lemma for barbed congruence in the π -calculus:
a solution to the third challenge of the Concurrent Calculi Formalization Benchmark

A Beluga formalization of the context lemma for barbed congruence in the π -calculus:
a solution to the third challenge of the Concurrent Calculi Formalization Benchmark

- Concurrent Calculi

A **Beluga formalization** of the context lemma for barbed congruence in the **π -calculus**:
a solution to the third challenge of the Concurrent Calculi Formalization Benchmark

- **Concurrent Calculi**
- **Interactive Theorem Proving**

A Beluga formalization of the context lemma for barbed congruence in the π -calculus:
a solution to the third challenge of the Concurrent Calculi Formalization Benchmark

- Concurrent Calculi
- Interactive Theorem Proving
- Benchmarks

A Beluga formalization of the context lemma for barbed congruence in the π -calculus:
a solution to the third challenge of the Concurrent Calculi Formalization Benchmark

- Concurrent Calculi
- Interactive Theorem Proving
- Benchmarks
- Coinduction

Roadmap

Part 1

Concurrent Calculi

π -calculus

Part 2

Interactive Theorem Proving

Beluga

Part 3

Benchmarks

Concurrent Calculi

Formalization Benchmark

Part 4

Coinduction

Barbed Congruence

Part 5

Results & Future Work

PART 1

Concurrent Calculi

π -Calculus

Concurrent systems

Concurrent systems: multiple processes interacting within a shared system

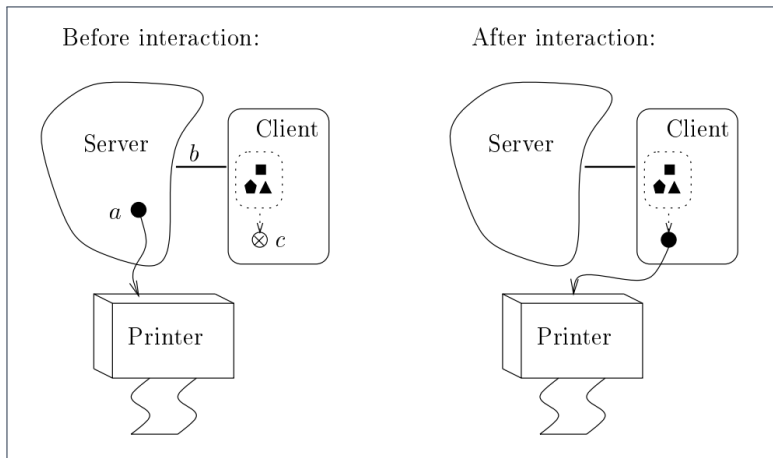


Concurrent calculi

Concurrent calculi: abstract models of concurrent systems

- ▶ CCS: *A Calculus of Communicating Systems*, Milner, 1980.
- ▶ π -**calculus**: *A Calculus of Mobile Processes*, Milner et al., 1992.

π -calculus: an example



π -calculus: syntax

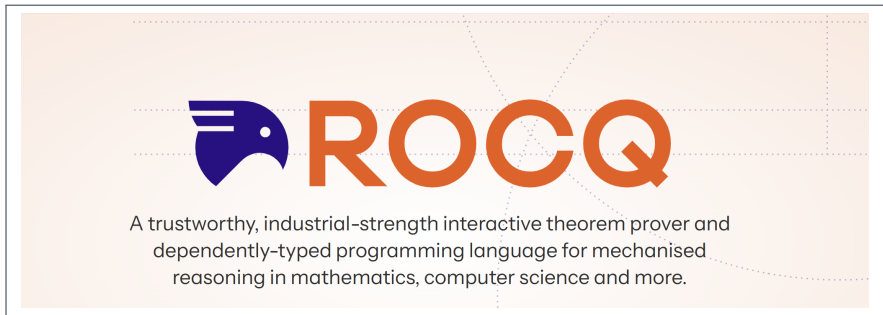
Processes: $P, P' ::= 0 \mid x(z).P \mid \bar{x}y.P \mid (P \mid P') \mid \nu z P \mid !P$

PART 2

Interactive Theorem Proving

Beluga

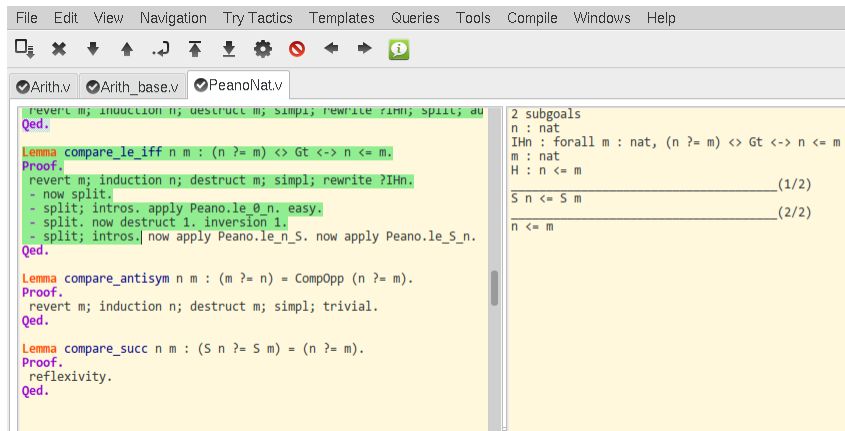
What are interactive theorem provers?



Computer programs for constructing machine-checked mathematical proofs

Scope: programming languages, compilers, software, mathematics, ...

What are their features?



- Interaction with the user (tactics, proof state, computation holes ...)

What are their features?

- ▶ Interaction with the user (tactics, proof state, computation holes ...)
- ▶ Automation

What are their features?

- ▶ Interaction with the user (tactics, proof state, computation holes ...)
- ▶ Automation
- ▶ Libraries

What are their features?

- ▶ Interaction with the user (tactics, proof state, computation holes ...)
- ▶ Automation
- ▶ Libraries
- ▶ Proofs checked by a trusted kernel

Why using them?

- ▶ Guarantee correctness of proofs

Why using them?

- ▶ Guarantee correctness of proofs
- ▶ Proof search and counterexamples search

Why using them?

- ▶ Guarantee correctness of proofs
- ▶ Proof search and counterexamples search
- ▶ Proofs with numerous cases, mostly not deep

Why using them?

- ▶ Guarantee correctness of proofs
- ▶ Proof search and counterexamples search
- ▶ Proofs with numerous cases, mostly not deep
- ▶ Increase comprehension of systems

The proof assistant Beluga



- ▶ Developed by the CompLogic group (Dr. Brigitte Pientka) at McGill University, Canada
- ▶ Designed for reasoning about formal systems: higher-order abstract syntax (HOAS), contexts as first-class objects
- ▶ Two-level system: representation level (LF), computation level
- ▶ Support for inductive and coinductive reasoning

PART 3

Benchmarks

Concurrent Calculi Formalization Benchmark

Benchmarks

Benchmarks are proposed to gauge and advance the state of the art:

Benchmark	Date	Primary focus
POPLMark	2005	Programming languages metatheory
List Machine	2006/12	Compiler correctness
ORBI	2015/17	Binders and contexts of assumptions
POPLMark Reloaded	2017/19	Logical-relations proofs
CCFB	2024	Concurrent calculi

How useful have benchmarks been?

How useful have benchmarks been?

Define **useful**

- ▶ Citations: some 460 for POPLMark, but not a good metric in general
- ▶ Abella & Beluga appeared after POPLMark, but have their foundations and motivations in work from the late 90's
- ▶ Implementation of features, bug detection, libraries for reasoning about binders

How useful have benchmarks been?

Define **useful**

- ▶ Citations: some 460 for POPLMark, but not a good metric in general
- ▶ Abella & Beluga appeared after POPLMark, but have their foundations and motivations in work from the late 90's
- ▶ Implementation of features, bug detection, libraries for reasoning about binders

A.M.:

"Benchmarks have been more successful as *coordination devices* than as requirements documents for proof assistants."

Concurrent calculi have been studied and formalized extensively over the past decades:

[illegible]

Concurrent Calculi Formalization Benchmark (2024)

Three benchmark problems to assess the state of the art and improve the mechanizations of concurrent calculi

Each problem addresses one common challenge occurring when formalizing them:

- ▶ Linearity & behavioural type systems
- ▶ Name passing & scope extrusion
- ▶ **Coinduction & infinite behaviour**

Third challenge: formalize a **context lemma** for barbed congruence in the π -calculus

PART 4

Coinduction

Barbed Congruence

Inductive datatypes

Finite structures can be represented via **inductive** datatypes.

Every element is obtained by finitely many **constructor** applications.

Example: finite lists of natural numbers

```
data list =  
  | nil: list  
  | cons: nat → list → list
```

The list $[1, 2, 3]$ can be represented as `cons 1 (cons 2 (cons 3 nil))`.

Coinductive datatypes

Potentially infinite structures can be represented via **coinductive** datatypes.
Every element is defined by specifying the results of applying **destructors** to it.

Example: streams of natural numbers

```
codata stream =  
  head: nat & tail: stream
```

The stream $0, 0, 0, \dots$ can be represented by the stream `zeros`, defined via
`head(zeros) = 0` and `tail(zeros) = zeros`.

Behavioral equivalences

In concurrent systems, processes can often run for an indefinite amount of time.
How can we tell whether two processes P and Q have the same behavior?

- ▶ At a given point, P and Q exhibit the same observables
- ▶ And this correspondence is preserved for their continuations

→ **Bisimulations** capture this idea coinductively: compare the current behavior, and require the same relation for the continuations.

Barbed bisimulation

Barbed bisimulation is an example of a coinductive notion of behavioral equivalence.

A symmetric relation $\mathcal{R}$ is a **strong barbed bisimulation** if $P \mathcal{R} Q$ implies:

1. $P \downarrow_x \Rightarrow Q \downarrow_x$
2. $P \downarrow_{\bar{x}} \Rightarrow Q \downarrow_{\bar{x}}$
3. $P \xrightarrow{\tau} P' \Rightarrow \exists Q'. Q \xrightarrow{\tau} Q' \wedge P' \mathcal{R} Q'$

Barbed bisimulation

Barbed bisimulation is an example of a coinductive notion of behavioral equivalence.

A symmetric relation $\mathcal{R}$ is a **strong barbed bisimulation** if $P \mathcal{R} Q$ implies:

1. $P \downarrow_x \Rightarrow Q \downarrow_x$
2. $P \downarrow_{\bar{x}} \Rightarrow Q \downarrow_{\bar{x}}$
3. $P \xrightarrow{\tau} P' \Rightarrow \exists Q'. Q \xrightarrow{\tau} Q' \wedge P' \mathcal{R} Q'$

In words, P and Q are barbed bisimilar if:

- ▶ They have the same current observable input/output capabilities (called *barbs*)
- ▶ Whenever one process performs an internal action, the other can match it, leading to processes that are related again.

PART 5

Results & Future Work

The context lemma, in a nutshell

Third challenge of CCFB: formalize a **context lemma** for barbed congruence in the π -calculus

Simply speaking, two processes are barbed congruent if:

- ▶ They are barbed bisimilar
- ▶ When placed in any context (a.k.a environment), they remain barbed bisimilar

The context lemma, in a nutshell

Third challenge of CCFB: formalize a **context lemma** for barbed congruence in the π -calculus

Simply speaking, two processes are barbed congruent if:

- ▶ They are barbed bisimilar
- ▶ When placed in any context (a.k.a environment), they remain barbed bisimilar

Difficult to prove: it requires bisimilarity to hold *for every context*

Context lemma: to prove barbed congruence, it is enough to consider a *restricted class of contexts*

Results and findings

- ▶ Formalization of one implication of the context lemma for barbed similarity in the π -calculus
- ▶ First solution of CCFB(3). This completes a trilogy of Beluga solutions of CCFB
- ▶ Formalization of contexts in the HOAS environment of Beluga, following [1]
- ▶ Discovery of a bug in the rules for process replication in the CCFB paper
- ▶ Proof of the equivalence of different variants of barbed bisimilarity

[1]: A. Lancelot, B. Accattoli & M. Vemclegs (2025): *Barendregt's Theory of the λ -Calculus, Refreshed and Formalized*

Lessons learned

- ▶ CCFB(3) is a valid case study for evaluating the mechanization of coinductive proofs. Not so much for up-to techniques
- ▶ HOAS and indexed codata $\rightarrow$ Beluga is well-suited for mechanizing binders and proofs by coinduction
- ▶ When dealing with coinduction and substitutions in Beluga, some checks (guardedness, totality, coverage) need to be done manually

Future work

- ▶ Journal paper collecting the three Beluga CCFB solutions
- ▶ Prove the converse implication of the context lemma
- ▶ Extend this result to weak barbed similarity
- ▶ Add process operators (e.g., *sum* and *match*) to the syntax

Thank you!

Any questions?

Lea Trogna · lea.trogna@studenti.unimi.it

Gabriele Cecilia · gcecilia@augusta.edu

Alberto Momigliano · alberto.momigliano@unimi.it

Paper: <https://doi.org/10.4204/EPTCS.448.1>

Code: github.com/LeaTrogna/formalizing-barbed-similarity-for-the-pi-calculus-in-beluga