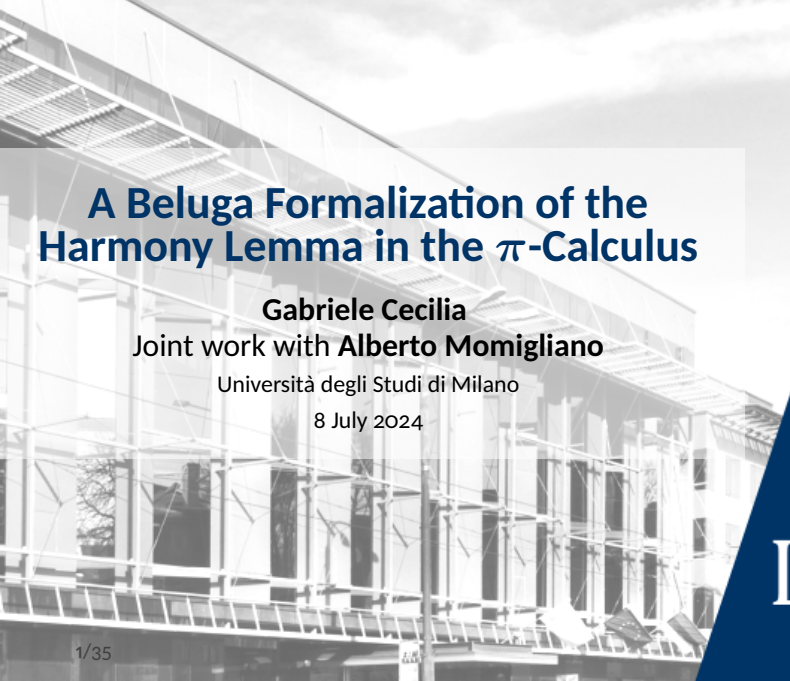


A Beluga Formalization of the Harmony Lemma in the π-Calculus

Gabriele Cecilia
Joint work with **Alberto Momigliano**
Università degli Studi di Milano

8 July 2024

LFMTP'24

Table of Contents

The POPLMark Challenge (2005)

"How close are we to a world where every paper on programming languages is accompanied by an electronic appendix with machine-checked proofs?"

The POPLMark Challenge (2005)

"How close are we to a world where every paper on programming languages is accompanied by an electronic appendix with machine-checked proofs?"

- Challenges about the formalization of the metatheory of programming languages
- Objectives: measure progress, find the best practices to address typical issues, improve proof assistants, stimulate collaboration
- New benchmarks: POPLMark Reloaded (2019), Concurrent Calculi Formalisation Benchmark (2024)

The Concurrent Calculi Formalisation Benchmark (2024)

Set of challenges about the formalization of concurrent systems

- Objectives: same as POPLMark, but focused on concurrency
- Three issues:
  i. Linearity
  ii. Scope extrusion $\rightarrow$ scope of restricted names may change over time
  iii. Coinductive reasoning

The Concurrent Calculi Formalisation Benchmark (2024)

Set of challenges about the formalization of concurrent systems

- Objectives: same as POPLMark, but focused on concurrency
- Three issues:
  i. Linearity
  ii. Scope extrusion $\rightarrow$ scope of restricted names may change over time
  iii. Coinductive reasoning

Second challenge of the CCFB: formalizing the Harmony Lemma for the π-calculus. Equivalence of two semantics which treat scope extrusion differently

Table of Contents

General features of the π-calculus

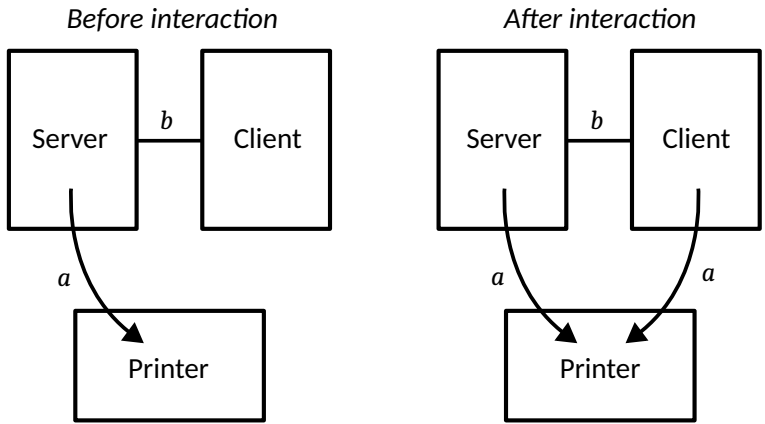
Process calculus: model for concurrent systems.

General features of the π-calculus

Process calculus: model for concurrent systems.

- Introduced by R.Milner, J. Parrow and D. Walker in 1992.
- Double nature of names: means of communication, data exchanged.
- It models processes whose interconnections change as they interact.

Example of process interaction:



Syntax

We assume the existence of a countably infinite set of names.

Definition

In the CCFB.2, the set of processes is defined by the following syntax:

$$P, Q ::= \mathbf{0} \mid x(y).P \mid \bar{x}y.P \mid (P \mid Q) \mid (\nu x)P$$

Syntax

We assume the existence of a countably infinite set of names.

Definition

In the CCFB.2, the set of processes is defined by the following syntax:

$$P, Q ::= \mathbf{0} \mid x(y).P \mid \bar{x}y.P \mid (P \mid Q) \mid (\nu x)P$$

Notation:
$\mathrm{fn}(P)$: set of free names in P
$\mathrm{bn}(P)$: set of bound names in P

Semantics

Process behaviour is defined through an operational semantics.

In the CCFB.2, two different approaches:

Semantics

Process behaviour is defined through an operational semantics.

In the CCFB.2, two different approaches:

- Reduction semantics: congruence + reduction
- Labelled Transition System (LTS) semantics: actions + transitions

Reduction Semantics

Definition

We define the congruence relation $\equiv$ as the smallest relation over processes, closed under compatibility and equivalence laws, satisfying the following axioms:

$$\textsc{Par-Assoc}\quad \frac{}{P \mid (Q \mid R) \equiv (P \mid Q) \mid R} \qquad \textsc{Par-Unit}\quad \frac{}{P \mid \mathbf{0} \equiv P} \qquad \textsc{Par-Comm}\quad \frac{}{P \mid Q \equiv Q \mid P}$$

$$\textsc{Sc-Ext-Zero}\quad \frac{}{(\nu x)\,\mathbf{0} \equiv \mathbf{0}} \qquad \textsc{Sc-Ext-Par}\quad \frac{x \notin \mathsf{fn}(Q)}{(\nu x)P \mid Q \equiv (\nu x)(P \mid Q)} \qquad \textsc{Sc-Ext-Res}\quad \frac{}{(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P}$$

Reduction Semantics

Definition

We define the reduction relation $\rightarrow$ as the smallest relation over processes satisfying the following rules:

$$\text{R-Com}\ \frac{}{\bar{x}y.P \mid x(z).Q \ \rightarrow\ P \mid Q\{y/z\}} \qquad \text{R-Par}\ \frac{P \rightarrow Q}{P \mid R \ \rightarrow\ Q \mid R}$$

$$\text{R-Res}\ \frac{P \rightarrow Q}{(\nu x)P \ \rightarrow\ (\nu x)Q} \qquad \text{R-Struct}\ \frac{P \equiv P' \qquad P' \rightarrow Q' \qquad Q' \equiv Q}{P \rightarrow Q}$$

Reduction Semantics

Definition

We define the reduction relation $\rightarrow$ as the smallest relation over processes satisfying the following rules:

$$\frac{}{\bar{x}y.P \mid x(z).Q \;\rightarrow\; P \mid Q\{y/z\}}\ \text{R-Com} \qquad \frac{P \rightarrow Q}{P \mid R \;\rightarrow\; Q \mid R}\ \text{R-Par}$$

$$\frac{P \rightarrow Q}{(\nu x)P \;\rightarrow\; (\nu x)Q}\ \text{R-Res} \qquad \frac{P \equiv P' \qquad P' \rightarrow Q' \qquad Q' \equiv Q}{P \rightarrow Q}\ \text{R-Struct}$$

Reduction Semantics

Definition

We define the reduction relation $\rightarrow$ as the smallest relation over processes satisfying the following rules:

$$\frac{}{\bar{x}y.P \mid x(z).Q \;\rightarrow\; P \mid Q\{y/z\}}\ \text{R-Com} \qquad \frac{P \rightarrow Q}{P \mid R \;\rightarrow\; Q \mid R}\ \text{R-Par}$$

$$\frac{P \rightarrow Q}{(\nu x)P \;\rightarrow\; (\nu x)Q}\ \text{R-Res} \qquad \frac{P \equiv P' \qquad P' \rightarrow Q' \qquad Q' \equiv Q}{P \rightarrow Q}\ \text{R-Struct}$$

LTS Semantics

Definition

The set of actions is defined by the following syntax:

$$\alpha ::= x(y) \mid \bar{x}y \mid \bar{x}(y) \mid \tau$$

LTS Semantics

Definition

The set of actions is defined by the following syntax:

$$\alpha ::= x(y) \mid \bar{x}y \mid \bar{x}(y) \mid \tau$$

Notation:
fn(α): set of free names in α
bn(α): set of bound names in α
n(α): set of names in α

LTS Semantics

Definition

We define the transition relation $\cdot \xrightarrow{\cdot} \cdot$ as the smallest relation which satisfies the following rules:

$$\text{S-In}\ \frac{}{x(z).P \xrightarrow{x(z)} P} \qquad \text{S-Out}\ \frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$$

$$\text{S-Par-L}\ \frac{P \xrightarrow{\alpha} P' \qquad \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\text{S-Com-L}\ \frac{P \xrightarrow{\bar{x}y} P' \qquad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{y/z\}} \qquad \text{S-Res}\ \frac{P \xrightarrow{\alpha} P' \qquad z \notin \mathsf{n}(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$$

$$\text{S-Open}\ \frac{P \xrightarrow{\bar{x}z} P' \qquad z \neq x}{(\nu z)P \xrightarrow{\bar{x}(z)} P'} \qquad \text{S-Close-L}\ \frac{P \xrightarrow{\bar{x}(z)} P' \qquad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} (\nu z)(P' \mid Q')}$$

LTS Semantics

Definition

We define the transition relation $\cdot \xrightarrow{\cdot} \cdot$ as the smallest relation which satisfies the following rules:

$$\text{S-In}\ \frac{}{x(z).P \xrightarrow{x(z)} P} \qquad \text{S-Out}\ \frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$$

$$\text{S-Par-L}\ \frac{P \xrightarrow{\alpha} P' \qquad \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\text{S-Com-L}\ \frac{P \xrightarrow{\bar{x}y} P' \qquad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{y/z\}} \qquad \text{S-Res}\ \frac{P \xrightarrow{\alpha} P' \qquad z \notin \mathsf{n}(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$$

$$\text{S-Open}\ \frac{P \xrightarrow{\bar{x}z} P' \qquad z \neq x}{(\nu z)P \xrightarrow{\bar{x}(z)} P'} \qquad \text{S-Close-L}\ \frac{P \xrightarrow{\bar{x}(z)} P' \qquad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} (\nu z)(P' \mid Q')}$$

LTS Semantics

Definition

We define the transition relation $\cdot \xrightarrow{\cdot} \cdot$ as the smallest relation which satisfies the following rules:

$$\text{S-In}\ \frac{}{x(z).P \xrightarrow{x(z)} P} \qquad \text{S-Out}\ \frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$$

$$\text{S-Par-L}\ \frac{P \xrightarrow{\alpha} P' \qquad \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\text{S-Com-L}\ \frac{P \xrightarrow{\bar{x}y} P' \qquad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{y/z\}} \qquad \text{S-Res}\ \frac{P \xrightarrow{\alpha} P' \qquad z \notin \mathsf{n}(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$$

$$\text{S-Open}\ \frac{P \xrightarrow{\bar{x}z} P' \qquad z \neq x}{(\nu z)P \xrightarrow{\bar{x}(z)} P'} \qquad \text{S-Close-L}\ \frac{P \xrightarrow{\bar{x}(z)} P' \qquad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} (\nu z)(P' \mid Q')}$$

LTS Semantics

Definition

We define the transition relation $\cdot \xrightarrow{\cdot} \cdot$ as the smallest relation which satisfies the following rules:

$$\text{S-In}\ \frac{}{x(z).P \xrightarrow{x(z)} P} \qquad \text{S-Out}\ \frac{}{\bar{x}y.P \xrightarrow{\bar{x}y} P}$$

$$\text{S-Par-L}\ \frac{P \xrightarrow{\alpha} P' \qquad \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\text{S-Com-L}\ \frac{P \xrightarrow{\bar{x}y} P' \qquad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{y/z\}} \qquad \text{S-Res}\ \frac{P \xrightarrow{\alpha} P' \qquad z \notin \mathsf{n}(\alpha)}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}$$

$$\text{S-Open}\ \frac{P \xrightarrow{\bar{x}z} P' \qquad z \neq x}{(\nu z)P \xrightarrow{\bar{x}(z)} P'} \qquad \text{S-Close-L}\ \frac{P \xrightarrow{\bar{x}(z)} P' \qquad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} (\nu z)(P' \mid Q')}$$

The Harmony Lemma

Harmony Lemma: the two semantics are equivalent

Theorem 1 (τ-transition implies reduction)

$P \xrightarrow{\tau} Q$ implies $P \to Q$.

Theorem 2 (Reduction implies τ-transition)

$P \to Q$ implies the existence of a Q' such that $P \xrightarrow{\tau} Q'$ and $Q \equiv Q'$.

The Harmony Lemma

Harmony Lemma: the two semantics are equivalent

"Rather than giving the whole proof, we explain the strategy and invite the reader to check some of the details" [Sangiorgi & Walker]

$\rightarrow$ complete the mathematical proof, filling out the details

Mathematical proof of the Harmony Lemma

Ingredients:

- 8 technical lemmas (substitutions, free/bound names)

Lemma (Names in output transitions)

If $P \xrightarrow{\bar{x}y} P'$, then $x, y \in \mathrm{fn}(P)$.

Mathematical proof of the Harmony Lemma

Ingredients:

- Some lemmas about rewriting

Lemma (Rewriting of processes involved in input transitions)

If $Q \xrightarrow{x(y)} Q'$ then there exists a finite (possibly empty) set of names $w_1, \ldots, w_n$ (with $x, y \neq w_i \; \forall i = 1, \ldots, n$) and two processes R, S such that

$Q \equiv (\nu w_1)\ldots(\nu w_n)(x(z).R \mid S)$ and $Q' \equiv (\nu w_1)\ldots(\nu w_n)(R \mid S)$.

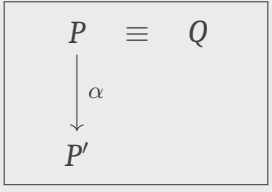
Mathematical proof of the Harmony Lemma

Ingredients:

- A congruence-as-bisimulation lemma

Lemma (Congruence is a bisimulation)

If $P \equiv Q$ and $P \xrightarrow{\alpha} P'$, then there exists a process Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv Q'$.



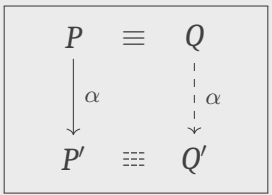
Mathematical proof of the Harmony Lemma

Ingredients:

- A congruence-as-bisimulation lemma

Lemma (Congruence is a bisimulation)

If $P \equiv Q$ and $P \xrightarrow{\alpha} P'$, then there exists a process Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \equiv Q'$.



Proofs are straightforward (sometimes long) inductions.

Table of Contents

Mechanization of the π-calculus

Complex aspects of the π-calculus mechanization:

- Binding constructs $\rightarrow$ α-equivalence, substitutions
- Scope extrusion $\rightarrow$ α-equivalence and variable conventions

Some previous π-calculus formalizations

Author, Year	Publication	Technique
Melham 94	A Mechanized Theory of the π-Calculus in HOL	Named syntax
Hirschkoff 97	A Full Formalisation of π-Calculus Theory in the Calculus of Constructions	De Bruijn indexes
Bengtson et al. 09	Formalizing the π-Calculus using Nominal Logic	Nominal techniques
Castro et al. 19	Engineering the Meta-Theory of Session Types	Locally nameless
Miller et al. 99	Foundational Aspects of Syntax	HOAS
Despeyroux 00	A Higher-Order Specification of the π-Calculus	HOAS
Honsell et al. 01	π-Calculus in (Co)Inductive Type Theory	HOAS

Why HOAS?

α-renaming and capture-avoiding substitutions are implemented by the meta-language

$\rightarrow$ Side conditions in definitions and technical lemmas are automatically achieved

Why HOAS?

α-renaming and capture-avoiding substitutions are implemented by the meta-language

$\rightarrow$ Side conditions in definitions and technical lemmas are automatically achieved

- Cleaner definitions and proofs
- The user can focus on the development of the target system

With other techniques, most of the effort is devoted to names handling (e.g. Hirschkoff, 75%)

Why Beluga?

Specifically designed for reasoning about formal systems

→ Encoding of object-level binding constructs through HOAS

→ Two-level system (LF-level, computation-level)

→ *Curry-Howard isomorphism*: proofs as recursive functional programs, propositions as types

Encoding of the Syntax

LF types for names and processes:

Names

```
LF names: type = ;
```

Processes

```
LF proc: type =
  | p_zero: proc
  | p_in: names → (names → proc) → proc
  | p_out: names → names → proc → proc
  | p_par: proc → proc → proc
  | p_res: (names → proc) → proc
;
```

$\mathbf{0}$	inactive
$x(y).P$	input
$\bar{x}y.P$	output
$P \mid Q$	parallel composition
$(\nu x)P$	restriction

Example (encoding of $(\nu x)P$): `p_res \x.(P x)`

Encoding of the Syntax

Terms are paired with contexts, containing assumptions. Contexts are classified through schemas

We need a context of the form "`x:names`":

Context declaration

```
schema ctx = names;
```

Encoding of the Syntax

Terms are paired with contexts, containing assumptions. Contexts are classified through schemas

We need a context of the form "`x:names`":

Context declaration

```
schema ctx = names;
```

Consequence: we can define *contextual* processes `[g ⊢ P]`, where `g` contains the free variables occurring in the open process `P`.

Encoding of the Reduction Semantics

Congruence and Reduction

```
LF cong: proc → proc → type =
  | sc_ext_par: cong ((p_res P) p_par Q) (p_res (\x.((P x) p_par Q)))
  ...
LF red: proc → proc → type =
  | r_res: ({x:names} red (P x) (Q x)) → red (p_res P) (p_res Q)
  ...
```

$$\textsc{Sc-Ext-Par}\ \frac{x \notin \mathsf{fn}(Q)}{(\nu x)P \mid Q \equiv (\nu x)(P \mid Q)}$$

- There is no side condition in rule `sc_ext_par`
- Universal quantification $\{\texttt{x:names}\}$ is used to descend into binders

Encoding of the LTS Semantics (Honsell et al.)

Actions and Transition

```
LF f_act: type =
  | f_tau: f_act
  | f_out: names → names → f_act
```

```
LF b_act: type =
  | b_in: names → b_act
  | b_out: names → b_act
```

- Two types for free and bound actions; no bound name in bound actions.

Encoding of the LTS Semantics (Honsell et al.)

Actions and Transition

```
LF f_act: type =
  | f_tau: f_act
  | f_out: names → names → f_act

LF b_act: type =
  | b_in: names → b_act
  | b_out: names → b_act

LF fstep: proc → f_act → proc → type =
  | fs_com1: fstep P (f_out X Y) P' → bstep Q (b_in X) Q'
    → fstep (P p_par Q) f_tau (P' p_par (Q' Y))
  | fs_close1: bstep P (b_out X) P' → bstep Q (b_in X) Q'
    → fstep (P p_par Q) f_tau (p_res \z.((P' z) p_par (Q' z))) ...
```

- The result of a free transition is a process.

Encoding of the LTS Semantics (Honsell et al.)

Actions and Transition

```
LF f_act: type =                          LF b_act: type =
  | f_tau: f_act                            | b_in: names → b_act
  | f_out: names → names → f_act             | b_out: names → b_act

LF fstep: proc → f_act → proc → type =
  | fs_com1: fstep P (f_out X Y) P' → bstep Q (b_in X) Q'
    → fstep (P p_par Q) f_tau (P' p_par (Q' Y))
  | fs_close1: bstep P (b_out X) P' → bstep Q (b_in X) Q'
    → fstep (P p_par Q) f_tau (p_res \z.((P' z) p_par (Q' z))) ...
and bstep: proc → b_act → (names → proc) → type =
  | bs_in: bstep (p_in X P) (b_in X) P  ...
```

- The result of a bound transition is a process abstraction.

Writing proofs in Beluga

Proofs in Beluga: total (recursive) functions

Proof term written by the user, without tactics

Encoding of the Harmony Lemma: technical lemmas

- Technical lemmas about substitutions and free/bound names:

HOAS $\rightarrow$ automatically achieved (except one)

Encoding of the Harmony Lemma: lemmas about rewriting

- Lemmas about rewriting:

New type family for existentials and sequences of binders, proof by induction

Encoding of the Harmony Lemma: lemmas about rewriting

Lemma (Rewriting of processes involved in input transitions)

If $Q \xrightarrow{x(y)} Q'$ then there exists a finite (possibly empty) set of names $w_1, \ldots, w_n$ (with $x, y \neq w_i \; \forall i = 1, \ldots, n$) and two processes R, S such that

$$Q \equiv (\nu w_1)\ldots(\nu w_n)(x(z).R \mid S) \quad \text{and} \quad Q' \equiv (\nu w_1)\ldots(\nu w_n)(R \mid S). \qquad (\star)$$

- Existentials and conjunctions $\rightarrow$ LF type families to encode them
- Sequences of binders $\rightarrow$ Inductive encoding (no binders / $(n+1)$ binders)

Encoding of the Harmony Lemma: lemmas about rewriting

Existential type for input rewriting

…there exist $w_1, \ldots, w_n, R, S$ such that one of the following holds:

i. $Q \equiv x(y).R \mid S$ and $Q' \equiv R \mid S$;

ii. $Q \equiv (\nu w)P$, $Q' \equiv (\nu w)P'$ and the congruences $(\star)$ hold for P and P'.

```
LF ex_inp_rew: proc → names → (names → proc) → type =
  | inp_base: Q cong ((p_in X R) p_par S)
    → ({y:names} (Q' y) cong ((R y) p_par S)) → ex_inp_rew Q X Q'
  | inp_ind: Q cong (p_res P) → ({y:names} (Q' y) cong (p_res (P' y)))
    → ({w:names} ex_inp_rew (P w) X \y.(P' y w)) → ex_inp_rew Q X Q'
```

Encoding of the Harmony Lemma: lemmas about rewriting

Types encoding existentials: analyzed by pattern matching through additional lemmas, proved by induction over their structure

In some cases, lexicographic induction on both arguments is required:

Auxiliary Lemma

```
rec fs_com1_impl_red: (g:ctx) [g ⊢ ex_fout_rew P1 X Y Q1]
  → [g ⊢ ex_inp_rew P2 X \x.Q2[..,x]]
  → [g ⊢ (P1 p_par P2) red (Q1 p_par Q2[..,Y])] = ...
```

→ Splitting the function in two, respectively decreasing on a single argument

Encoding of Harmony Lemma: congruence-as-bisimilarity lemma

- Congruence-as-bisimilarity lemma:
  1. We need a technical lemma

Strengthening Lemma

If $\Gamma, \texttt{x:names} \vdash P \xrightarrow{\alpha_x} Q_x$, then there are α', Q' such that $\alpha_x = \alpha'$, $Q_x = Q'$ and $\Gamma \vdash P \xrightarrow{\alpha'} Q'$

```
rec strengthen_fstep: (g:ctx) {F:[g,x:names ⊢ fstep P[..] A Q]}
  → ex_str_fstep [g,x:names ⊢ F] = ...
```

Encoding of Harmony Lemma: congruence-as-bisimilarity lemma

- Congruence-as-bisimilarity lemma:
  2. It has *two* symmetrical assertions regarding *both* free and bound transitions
     $\rightarrow$ Encoded through four mutual recursive functions (long but straightforward proof)

Lemma (Congruence is a bisimilarity)

```
rec cong_fstepleft_impl_fstepright: (g:ctx) [g ⊢ P cong Q]
  → [g ⊢ fstep P A P'] → [g ⊢ ex_fstepcong P Q A P'] = ...
```

Encoding of the Harmony Lemma: Theorems 1 and 2

Theorem 1 (τ-transition implies reduction)

```
rec fstep_impl_red: (g:ctx) [g ⊢ fstep P f_tau Q] → [g ⊢ P red Q] =
fn f ⇒ case f of
   | [g ⊢ fs_com1 F1 B1] ⇒
     let [g ⊢ D1] = fs_out_rew [g ⊢ _] [g ⊢ F1] in
     let [g ⊢ D2] = bs_in_rew [g ⊢ _] [g ⊢ B1] in
     let [g ⊢ R] = fs_com1_impl_red [g ⊢ D1] [g ⊢ D2] in [g ⊢ R] ...
```

Theorem 2 (Reduction implies τ-transition)

```
rec red_impl_fstepcong: (g:ctx) [g ⊢ P red Q]
  → [g ⊢ ex_fstepcong P P f_tau Q] =
/ total r (red_impl_fstepcong _ _ _ r) /
fn r ⇒ let [g ⊢ D1] = red_impl_red_rew r in
       let [g ⊢ D2] = red_rew_impl_fstepcong [g ⊢ D1] in [g ⊢ D2]
```

Table of Contents

Evaluation

Some numbers:

	Informal proof	Beluga formalization
Proof size	∼30 pages	∼700 lines
Technical lemmas	8	1
Theorems	2	2
Main lemmas	5	5

Evaluation

Some numbers:

	Informal proof	Beluga formalization
Proof size	∼30 pages	∼700 lines
Technical lemmas	8	1
Theorems	2	2
Main lemmas	5	5

- Concise formalization: HOAS + Beluga

Evaluation

Some numbers:

	Informal proof	Beluga formalization
Proof size	∼30 pages	∼700 lines
Technical lemmas	8	1
Theorems	2	2
Main lemmas	5	5

- One-to-one correspondence between formal and informal proof

Evaluation

Some numbers:

	Informal proof	Beluga formalization
Proof size	∼30 pages	∼700 lines
Technical lemmas	8	1
Theorems	2	2
Main lemmas	5	5

• There are additional lemmas in the formalization, due to the current state of Beluga (e.g. lack of construct for existentials) and corresponding to parts of the informal lemmas

Evaluation

Positive aspects of Beluga:

- HOAS $\rightarrow$ less technical details
- Almost uneventful formalization process $\rightarrow$ suitable environment for this system
- Reliable totality checker, despite heavy use of mutual recursion

Negative aspects of Beluga:

- Lack of a construct for existentials and conjunctions

Conclusions

Results:

- First formal and informal proof about semantics equivalence
- Development of techniques to encode specific constructs (sequences of binders, lexicographic induction)
- Contribution to the Concurrent Calculi Formalisation Benchmark

Future Work

- Proving semantics equivalence for an extended version of the π-calculus
- Experimenting automation (Harpoon)
- Providing a Beluga solution for all the CCFB problems

Thank you for listening!
Any questions?