Barbed Similarity for the π-Calculus in Beluga: A Case Study in Coinductive Reasoning

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

Gabriele Cecilia
School of Computer & Cyber Sciences,
Augusta University, Augusta, USA

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

We formalize strong barbed similarity for the π-calculus in the Beluga proof assistant, completing a line of work addressing the Concurrent Calculi Formalization Benchmark. By extending previous developments to include replication, we give a coinductive encoding of behavioral equivalence based on barbs and internal actions. Using Beluga's copattern-based coinduction, we obtain concise and compositional proofs, including compatibility properties and a context lemma characterizing barbed precongruence. The case study demonstrates the effectiveness of combining HOAS and coinductive reasoning for mechanizing concurrent calculi.

1 Introduction

One of the reasons why people flock to cinemas to watch sequels and "rebooted" movies is that they know what they are getting into: they are familiar with the characters and the basic setup. Keeping with the cinematographic metaphor, we will cut to the chase and assume that the reader is already on board with the idea that the best assurance of the correctness of properties of a formal calculus is a machine-checked proof. In fact, this is becoming the norm in the semantics of programming languages; it is less established when *concurrency* is concerned, although the latter is of overwhelming importance in modern computing.

Enter the Concurrent Calculi Formalization Benchmark [4] (CCFB), which provides a suite of problems designed to evaluate the formalization of concurrent and distributed language models, in particular process calculi. Following the methodology of the POPLMark challenge, CCFB aims to assess current mechanization techniques, identify optimal formalization patterns, and ideally drive improvements in proof assistant tooling. The CCFB framework structures this evaluation around three orthogonal challenges: the management of linear resources, the handling of scope extrusion, and the implementation of coinductive proof techniques. For the latter, the challenge is to prove the "context lemma" for *strong barbed bisimilarity*, namely that this notion of bisimilarity can be turned into a congruence by making it sensitive to substitution and parallel composition.

Induction is one of the great success stories of proof assistants: although systems differ in the details of their foundations and automation, inductive definitions and proofs by induction are supported in essentially comparable ways across the board. Coinduction has had a less uniform history; for a survey, we refer to [17]. The earliest systematic approach, due to Paulson, treats coinductive predicates as greatest fixed points of monotone operators, from which the corresponding coinduction principles are derived.

A different tradition, familiar from Rocq/Coq, is based on guarded corecursion: coinductive objects are introduced by constructors, and recursive calls must occur under such constructors to ensure productivity. This discipline is often brittle in proof developments, especially when coinductive arguments are

To appear in EPTCS.

nested, combined with induction, or hidden behind auxiliary lemmas. Agda has explored several alternatives over time, from delays to sized types and observational presentations of codata to guarded type theories with the "later" modality [16]. Similarly, Beluga has embraced sized types and copatterns [19]. These developments are particularly relevant for mechanized concurrency, where bisimulations and up-to techniques routinely require coinductive proofs that are interleaved with substantial inductive reasoning about syntax, transitions, substitutions, and contexts.

The present paper is the conclusion of the Beluga "trilogy" of solutions to CCFB, initiated with [23] — featuring, among other contributions, a solution to the linearity challenge — and followed by [5] with the mechanization of the Harmony Lemma. As in any good sequel, we borrow from the latter paper the implementation of the syntax and the LTS-based operational semantics of the π-calculus, while crucially extending it with *replication*. After all, this is what makes behavioral equivalence interesting. And interesting it turns out to be, as there is a twist in this movie: the rule governing replication as proposed in [4] is insufficient to establish the main result of the challenge. In fact, under this rule, structural congruence is not included in strong barbed similarity. This is problematic, because structural congruence is the baseline syntactic equivalence between processes and the context lemma relies crucially on this specific inclusion.

While this lapse is mildly embarrassing, given the singleton intersection between the authors of the Benchmark and of the present paper, it is once more a reaffirmation of the usefulness of mechanizations in proof assistants.

We will assume familiarity with the basic notions of the π-calculus as in the first chapters of [18], as well as a working knowledge of Beluga, both of its syntax and of its approach to proof checking. In particular we will only briefly touch upon the way Beluga handles coinduction via observations and refer the reader to [19] for the theory and to [15] for an application.

In the following, the statements of informal lemmas, theorems and proofs are hyperlinked via the accompanying cute icon  to their formalization in the repository:

`https://github.com/LeaTrogni/formalizing-barbed-similarity-for-the-pi-calculus-in-beluga`

As a final note, we have cut one corner: we have concentrated on similarity, precongruence, etc., rather than bisimilarity and congruence. Extending the results to the symmetric case would only duplicate the code with no new insights and could be left to automation, perhaps a coding agent.

2 The π-Calculus and its Operational Semantics

To make the paper self-contained, we present the main definitions of the syntax and the labelled transition system (LTS) of the fragment of the π-calculus under study. For more details about the informal definitions, we refer the reader to [18].

2.1 Syntax

We follow the definitions of CCFB3 excluding sums and (mis)match, while we diverge from it by allowing the calculus to be name-passing rather than value-passing. That original separation of concerns is not a simplification in the HOAS approach, but just a quirk.

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

Recall that the input prefix $x(y).P$ and the restriction $(\nu y)P$ both bind the name y in P, while any other occurrence of names in a process is free. Accordingly, we write $\mathsf{fn}(P)$ and $\mathsf{bn}(P)$ for the sets of free and bound names occurring in a process.

Beluga is based on a two-level system, with the LF level for representing data and a computation level for reasoning about it via recursion and pattern matching. In the encoding, names are defined by an LF type `names` without any constructor, together with a context schema that allows names to occur in other (open) LF objects. Since we define no other context schema, all the variables in open terms will have type `names`. Processes are LF objects, encoded using (weak) HOAS for input and restriction, as well known and detailed in [5]; their encoding is displayed in Fig. 1. Throughout the development, lowercase identifiers denote LF types, while those beginning with an uppercase letter denote computation-level data.

```
LF names: type =;

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
  | p_rep: proc → proc; --infix p_par 11 left.

schema ctx = names;
```

Figure 1: Encoding of names and processes.

2.2 Operational Semantics

Actions include inputs $x(y)$, free and bound outputs $\bar{x}y$ and $\bar{x}(y)$, and internal communications τ. In inputs and bound outputs, the occurrences of the name y are bound, while all other name occurrences in actions are free; from this, we obtain the standard notions of free names, bound names and names occurring in an action α, respectively denoted as $\mathsf{fn}(\alpha)$, $\mathsf{bn}(\alpha)$ and $\mathsf{n}(\alpha)$.

Our LTS, displayed in Fig. 2, presents some differences from the one in CCFB3. First, and less importantly, we have adopted *late* instead of *early* semantics: not only can they be proven equivalent in terms of transitions, see [5] for the mechanization of this folk result, but they induce the same strong barbed similarity, as established by Lemma 3.1 below and the discussion that follows. Secondly, as we teased in the introduction, we add two replication rules for communication to the LTS presented in CCFB3, since without them structural congruence is not included in strong barbed similarity: in the accompanying formalization we exhibit a pair of structurally congruent processes that are not barbed similar. This conflicts with the intended role of structural congruence as the strongest equivalence between processes, and the proof of the context lemma relies on this specific inclusion. The adoption of the two replication rules for communication is not arbitrary; after all, our LTS follows that in Sangiorgi and Walker's textbook, modulo late semantics. A popular alternative, found e.g. in [8, 3], uses a single rule to generate arbitrary copies of the replicated process. We refer to Sangiorgi and Walker for a discussion of the trade-offs between these formulations; the correspondence is understood modulo structural congruence.

Following the previous formalizations in [13, 8, 21], in Beluga we separate free steps, where the action does not bind any variable, from bound steps, where the action has a bound variable. Since some rules involve both free and bound actions at the same time, the LF types for free and bound steps are mutually defined. This approach does duplicate some rules, but gets rid of *all* provisos on free and bound occurrences, which are now taken care of by dependencies (or lack thereof) in second-order process variables.

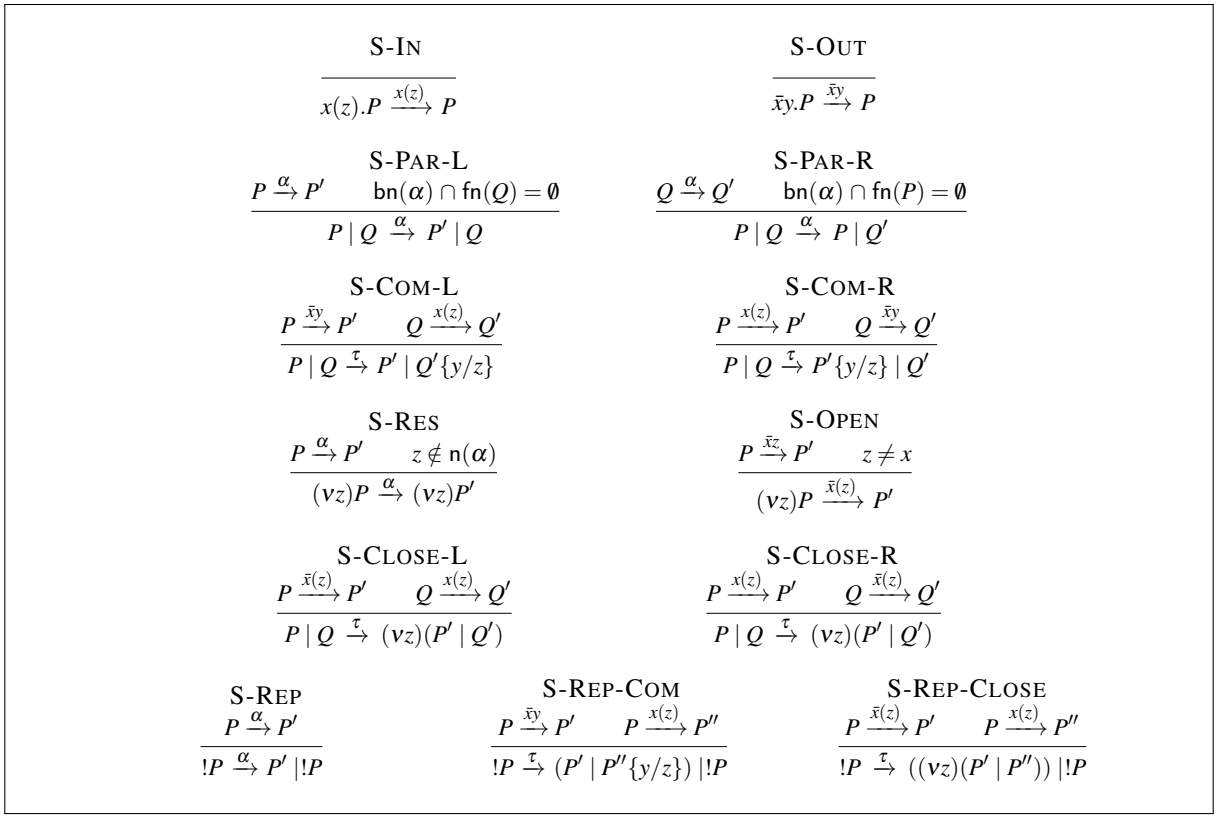

Figure 2: Transition rules.

2.3 Process contexts and structural congruence

The notion of *(pre)congruence* is of paramount importance in the π-calculus: both in the reduction semantics, through *structural* congruence, and in the study of behavioral equivalence. Informally, a congruence is an equivalence relation that preserves the behavior of the constructors. Often, this is described in textbooks (e.g. [18]) via the notion of *process context*: a process where a (non-degenerate) occurrence of $\mathbf{0}$ is replaced by a hole. Given a context C and a process P, $C[P]$ denotes the process obtained by replacing the hole in C with P. Then, a process *pre*congruence is a binary relation $\mathscr{R}$ which is a preorder such that if $P \mathscr{R} Q$, then, for each context C, $C[P] \mathscr{R} C[Q]$.

There is however a peculiarity: since the goal of contexts is to observe how processes behave in every possible environment, they need to be able to capture the free variables in the processes they are filled with. In other words, contexts do not respect α-equivalence.

This brings in some additional issues with respect to a mechanization, in particular in a HOAS setting, where such entities are a non-starter [1]. A workaround is to close the relation under *compatibility* ([12,7]), as described by the rules in our case at the top of Fig. 3. Concretely, when we say that *structural congruence* is the smallest congruence satisfying the axioms in the bottom of Fig. 3, we are instantiating $\mathscr{R}$ with $\equiv$.

This is indeed the way structural congruence was formalized as an LF type `cong` in [5], which we have extended by adding the compatibility and unfolding laws for replication.

[1] HOAS is indeed compatible with weaker notions such as *evaluation* contexts, i.e., those that do not cross a binder.

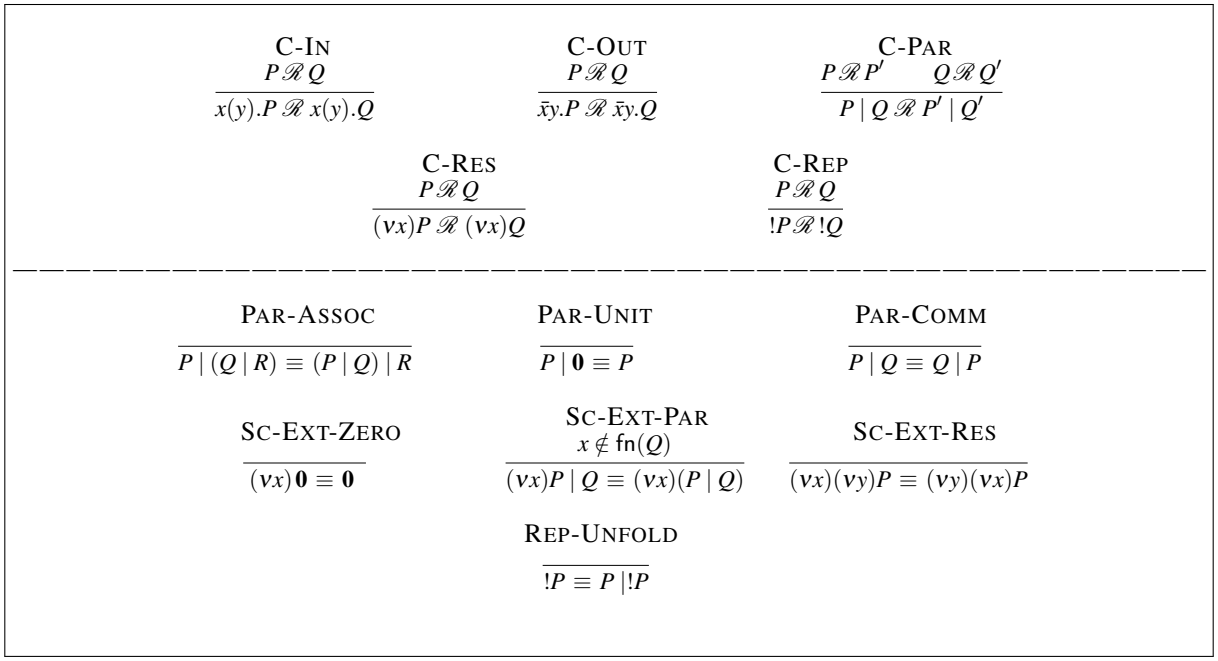

Figure 3: Compatibility rules and structural congruence axioms.

There are cases where going the compatibility route is inconvenient, if not plain inadequate, one being the definition of *strong barbed (pre)congruence*, as we shall see later on, and where contexts must be encoded. In a breakthrough, the authors of [11] introduced a clever trick to make contexts compatible (no pun intended) with a HOAS encoding, by means of a quaternary relation that implicitly captures the essence of a context:

$$\{(P,P',Q,Q') \mid P' = C[P],\ Q' = C[Q] \text{ for some context } C\}$$

Unlike in the Abella setting where this relation was introduced, a Beluga encoding is rather subtle as the LF contexts must be made explicit. In fact, in general $\mathsf{fn}(C[P]) \neq \mathsf{fn}(P)$, since the binders in C may capture names that occur free in P, and C may also introduce names that are fresh for P; while the latter can be avoided by working in an LF context that already contains all free variables of P and C, the former inevitably leads to a mismatch between the two LF contexts. Therefore, the formalization of this relation cannot be an LF predicate, where all the related terms are defined in an implicit context. Instead, we define the inductive type `PCtx` in Fig. 4, where the context of the second and fourth processes is a prefix of the context of the first and third processes.

```
inductive PCtx: (g:ctx)(g':ctx)[g' ⊢ proc] → [g ⊢ proc] → [g' ⊢ proc] → [g ⊢ proc] → ctype =
  | pidM: PCtx [g ⊢ P] [g ⊢ P] [g ⊢ Q] [g ⊢ Q]
  | pinM: PCtx [g' ⊢ P] [g,x:names ⊢ CP] [g' ⊢ Q] [g,x:names ⊢ CQ]
       → PCtx [g' ⊢ P] [g ⊢ p_in X (\x.CP)] [g' ⊢ Q] [g ⊢ p_in X (\x.CQ)]
  | poutM: PCtx [g' ⊢ P] [g ⊢ CP] [g' ⊢ Q] [g ⊢ CQ]
       → PCtx [g' ⊢ P] [g ⊢ p_out X Y CP] [g' ⊢ Q] [g ⊢ p_out X Y CQ]
  ...
```

Figure 4: Excerpt from encoding of contexts.

Structural congruence can alternatively be defined as a contextual equivalence via the inductive type `Cong` in Fig. 5, that uses the type `PCtx` in the *context closure* clause.

```
inductive Cong: (g:ctx) [g ⊢ proc] → [g ⊢ proc] → ctype =
  ...
% Replication unfolding
  | Rep_unfold: Cong [g ⊢ (p_rep P)] [g ⊢ (P p_par (p_rep P))]
% Context closure
  | C_ctx: Cong [g' ⊢ P] [g' ⊢ Q] → PCtx [g' ⊢ P] [g ⊢ CP] [g' ⊢ Q] [g ⊢ CQ]
        → Cong [g ⊢ CP] [g ⊢ CQ]
% Equivalence Relation Laws
  | C_ref: Cong [g ⊢ P] [g ⊢ P]
  | C_sym: Cong [g ⊢ P] [g ⊢ Q] → Cong [g ⊢ Q] [g ⊢ P]
  | C_trans: Cong [g ⊢ P] [g ⊢ Q] → Cong [g ⊢ Q] [g ⊢ R] → Cong [g ⊢ P] [g ⊢ R]
;
```

Figure 5: Excerpt from the inductive definition of structural congruence.

2.4 Preliminary lemmas

We start by establishing some basic properties of the LTS and of process contexts. The former ones can be found in [18], modulo our choice of late semantics, while the latter ones, adapted from [11], stem from the implementation of process contexts.

Lemma 2.1 (Properties of the LTS)**.**

1. *If* $P \xrightarrow{\bar{x}y} P'$ *and* z *is a name, then either* $z = y$ *and* $\nu z P \xrightarrow{\bar{x}(z)} P'$ *or* $\nu z P \xrightarrow{\bar{x}y} \nu z P'$.
2. *If* $S \mid !P \xrightarrow{\tau} R$, *then there is* Q *such that* $S \mid (P \mid P) \xrightarrow{\tau} Q$ *and* $R \equiv Q \mid !P$.
3. *If* $x \notin \mathrm{fn}(P)$ *and* $P \xrightarrow{\alpha} P'$, *then* $x \notin (\mathrm{fn}(\alpha) \cup \mathtt{fn}(P'))$.
4. *If* $P \equiv Q$, *then*
   - *if* $P \xrightarrow{\alpha} P'$, *then there is* Q' *such that* $Q \xrightarrow{\alpha} Q'$ *and* $P' \equiv Q'$;
   - *if* $Q \xrightarrow{\alpha} Q'$, *then there is* P' *such that* $P \xrightarrow{\alpha} P'$ *and* $P' \equiv Q'$.

Proof. (Sketch) The first two statements are proven by a straightforward induction on the derivation of the transition in the hypothesis. The third one is split into two lemmas, separating free and bound actions, which are proven by a mutual induction on the derivation of the transition in the hypothesis. The last one is split into four lemmas (it requires to prove two different theses and in both we need to distinguish free and bound actions), which are proven by a mutual induction on the derivation of the congruence in the hypothesis. We note that the third and the fourth results are extensions of those in [5]. □

Now for properties of process contexts: some of them would be obvious with the on-paper definition of contexts, but are not trivial in the `PCtx` formalization. We recall that $((P,C_P),(Q,C_Q))$ means that there is a context C such that $C[P] = C_P$ and $C[Q] = C_Q$.

Lemma 2.2 (Properties of contexts)**.**

1. *(Symmetry) If* $((P,C_P),(Q,C_Q))$, *then* $((Q,C_Q),(P,C_P))$.

2. *(Transitivity) If* $((P,C_P),(R,C_R))$*, then for all* Q *there is* C_Q *such that* $((P,C_P),(Q,C_Q))$ *and* $((Q,C_Q),(R,C_R))$.
3. *(Functionality) If* $((P,C_P),(P,C'_P))$*, then* $C_P = C'_P$.
4. *(Context composition) If* $((P,C_P),(Q,C_Q))$ *and* $((C_P,C_{C_P}),(C_Q,C_{C_Q}))$*, then* $((P,C_{C_P}),(Q,C_{C_Q}))$.

Proof. All proofs are by induction on the derivation of the context relation in the hypothesis. □

As is well known [12], a binary relation on processes constitutes a contextual equivalence if and only if it is a compatible equivalence. Because Beluga's logic lacks support for higher-order predicates, this meta-theorem cannot be generalized; rather, the equivalence must be formalized independently for each specific relation.

Lemma 2.3. *Structural congruence can be characterized either via compatibility or by context closure.*

Proof. (Sketch) Both inclusions are proved by induction on the derivation of the congruence rule in the hypothesis. The inclusion of the compatible relation in the contextual one first requires to prove that the compatible relation is contextually closed too. □

The LF definition is more convenient for discussing the properties of the LTS, as in [5], while the inductive one is used later to show the inclusion of structural congruence in another precongruence defined via contexts.

3 Behavioral Equivalences

Intuitively, two processes can be considered equivalent when they exhibit the same behavior. However, the level of detail at which behaviors can be distinguished depends on the chosen observational granularity; this is where a variety of bisimilarity notions come into play. In CCFB3, the authors focus on *strong barbed bisimilarity*, which relates processes that can match internal (τ) transitions and preserve the same *barbs*, i.e. the ability to perform input or output on a given channel.

More precisely, the barb, or observability predicate, $P \downarrow_x$ (resp. $P \downarrow_{\overline{x}}$) holds if a process P can perform an input action with subject x (resp. an output action with subject $\overline{x}$). Following [18], this notion can be formalized in two equivalent ways, based either on the structure of the process P or on the (late) LTS:

i) $P \downarrow_x$ iff $P \equiv \nu z_1 \ldots \nu z_n (x(y).Q \mid R)$ for some $y, z_1, \ldots, z_n, Q, R$, with $x \notin \{z_1, \ldots, z_n\}$. Analogously, $P \downarrow_{\overline{x}}$ iff $P \equiv \nu z_1 \ldots \nu z_n (\overline{x}y.Q \mid R)$ for some $y, z_1, \ldots, z_n, Q, R$, with $x \notin \{z_1, \ldots, z_n\}$.
The telescopes, i.e. n-ary sequences of binders, appearing in this definition of barb are encoded in Beluga by two types `barb_in_rew` and `barb_out_rew`, that build the sequence of restrictions incrementally. Then, the two types encoding barbs identify processes congruent to such telescopes.

ii) $P \downarrow_x$ iff $P \xrightarrow{x(z)} P'$ for some z, P', and similarly $P \downarrow_{\overline{x}}$ iff $P \xrightarrow{\overline{x}y} P'$ or $P \xrightarrow{\overline{x}(z)} P'$ for some y, z, P'.
We formalize this at the meta level, but it could have been formulated as an LF type as easily. Since the LTS presents one input label and two output labels, `Barb_in` has one constructor, while `Barb_out` has two.

Definitions i) and ii) are equivalent, in the sense given by the following lemma. For brevity, we omit some auxiliary technical lemmas and refer to the repository for full details. 

Lemma 3.1. *Let P be a process, x be a name. The following are equivalent:*

1. $P \equiv \nu z_1 \ldots \nu z_n (x(y).Q \mid R)$ *for some* $y, z_1, \ldots, z_n, Q, R$, *with* $x \notin \{z_1, \ldots, z_n\}$;
2. $P \xrightarrow{x(z)} P'$ *for some* z, P'. 

Similarly, the following are equivalent:

1. $P \equiv \nu z_1 \ldots \nu z_n (\bar{x}y.Q \mid R)$ *for some* $y, z_1, \ldots, z_n, Q, R$, *with* $x \notin \{z_1, \ldots, z_n\}$;
2. $P \xrightarrow{\bar{x}y} P'$ *or* $P \xrightarrow{\bar{x}(z)} P'$ *for some* y, z, P'. 

Proof. (Sketch) The proof that i) implies ii) is a straightforward induction on the rewriting definitions. The converse proceeds by structural induction on P, followed by an inversion on the observable transition that stems from P; some subcases are handled by the omitted auxiliary lemmas. □

Being invariant w.r.t. the underlying LTS, definition i) can be adopted even in the setting of reduction semantics; however, definition ii) has the benefit of being more amenable to mechanized proofs without the need of algebraic rewrites. For this reason, we use the latter in the rest of the development.

A significant property of barbs is that they are preserved by contexts:

Lemma 3.2.

1. *If* $P \downarrow_x$ *implies* $Q \downarrow_x$ *and* $((P, C_P), (Q, C_Q))$, *then* $C_P \downarrow_x$ *implies* $C_Q \downarrow_x$. 
2. *If* $P \downarrow_{\bar{x}}$ *implies* $Q \downarrow_{\bar{x}}$ *and* $((P, C_P), (Q, C_Q))$, *then* $C_P \downarrow_{\bar{x}}$ *implies* $C_Q \downarrow_{\bar{x}}$. 

Proof. By induction on the derivation of the context relation. □

We can now give the definition of barbed simulation. A binary relation on processes $\mathscr{R}$ is a *strong barbed simulation* if the following conditions hold:

1. $\mathscr{R}$ is *barb preserving*: if $P \mathscr{R} Q$ and $P \downarrow_x$, then $Q \downarrow_x$; likewise, if $P \mathscr{R} Q$ and $P \downarrow_{\bar{x}}$, then $Q \downarrow_{\bar{x}}$.
2. $\mathscr{R}$ is a *reduction simulation*: if $P \mathscr{R} Q$ and $P \xrightarrow{\tau} P'$, then there is Q' such that $Q \xrightarrow{\tau} Q'$ and $P' \mathscr{R} Q'$.

The union of all barbed simulations is called *strong barbed similarity* and will be denoted by $\overset{\bullet}{\leq}_B$.

As explained above, barbs are not based on a specific LTS, and, as proven in [5], albeit without the replication operator, early and late semantics provide the same τ actions. Thus, our definition of strong barbed similarity turns out to be equivalent to the one based on the early semantics.

(Bi)simulations are perhaps the best known instance of coinductively defined relations. In calculi where the transition relation is well-founded — e.g., those where each transition produces a structurally smaller process, ensuring that every computation trace is finite — (bi)similarity can equivalently be characterized inductively. This is not the case in our fragment of the π-calculus, where infinite behaviors arise due to replication: thus, a genuinely coinductive treatment becomes necessary.

To our rescue, Beluga supports coinductive reasoning in a very flexible way [19]. Coinductive types are defined dually to inductive types: instead of constructors, they are characterized by destructors, or *observations*. More specifically, while an inductive type is defined by inference rules whose conclusion is an instance of the type, a coinductive type is defined by inference rules where an instance of the type is the premise. Beluga separates this premise from the rest of the rule by using `::`.

Reasoning on coinductive objects proceeds via *copattern matching*. By the Curry-Howard correspondence, proofs by (co)induction are encoded as total (co)recursive functions. In the coinductive case, totality requires coverage of all observations, while productivity ensures that each corecursive call is guarded by an observation; in the (current) absence of a productivity checker in Beluga, this condition must be verified manually. Observations of a coinductive object `c` are accessed using the syntax `c.observation_name`, mirroring field access in mainstream programming languages.

As an example of a coinductive definition, we can look at the formalization of strong barbed similarity in Fig. 6, which presents three observations, corresponding to the input barbs, the output barbs and the τ transitions. Recall that Beluga does not provide constructs for existential quantification, conjunction, or disjunction, so these must be encoded as inductive types.

```
coinductive BarbSim : (g:ctx) [g ⊢ proc] → [g ⊢ proc] → ctype =
  | (BarbSim_barb_in : BarbSim [g ⊢ P] [g ⊢ Q])
              :: Barb_in [g ⊢ P] [g ⊢ X] → Barb_in [g ⊢ Q] [g ⊢ X]
  | (BarbSim_barb_out : BarbSim [g ⊢ P] [g ⊢ Q])
              :: Barb_out [g ⊢ P] [g ⊢ X] → Barb_out [g ⊢ Q] [g ⊢ X]
  | (BarbSim_tau : BarbSim [g ⊢ P] [g ⊢ Q])
              :: [g ⊢ fstep P f_tau P'] → Ex_sim_barb [g ⊢ P] [g ⊢ Q] [g ⊢ P']

and inductive Ex_sim_barb : (g:ctx) [g ⊢ proc] → [g ⊢ proc] → [g ⊢ proc] → ctype =
  | Ex_sim_barb_def : [g ⊢ fstep Q f_tau Q'] → BarbSim [g ⊢ P'] [g ⊢ Q']
                  → Ex_sim_barb [g ⊢ P] [g ⊢ Q] [g ⊢ P']
;
```

Figure 6: Coinductive definition of strong barbed similarity.

Lemma 3.3 (Properties of strong barbed similarity)**.** *Strong barbed similarity*

1. *is a preorder;*
2. *is preserved by input prefix, output prefix and restriction;*
3. *includes structural congruence.*

Proof.

1. By a straightforward coinduction, as in [15].
2. To illustrate how Beluga implements a coinductive proof, we detail the restriction case. On paper, we would prove that $\{(\nu z P, \nu z Q) \mid P \dot{\leq}_B Q\}$ is a strong barbed simulation and therefore is included in $\dot{\leq}_B$. In Beluga, this is realized by the recursive function in Fig. 7, whose signature encodes the statement. The proof distinguishes cases on the possible observations — note the copattern syntax reminiscent of record selection:
   - In the barbs cases we employ Lemma 3.2, by which if the barbs of P are included in the barbs of Q, then for each context C (and in particular $\nu z[\_]$), the barbs of $C[P]$ are included in the barbs of $C[Q]$.
   - More interestingly, in the transition case we invert on the restriction rule for a τ action from P to some P' (the first `let`). Then, using the hypothesis $P \dot{\leq}_B Q$ (`BarbSim`), we can find a matching action from Q to Q', where $P' \dot{\leq}_B Q'$. Finally, we can corecursively call the theorem

on $P' \dot{\leq}_B Q'$ and, by applying the restriction rule forward, obtain the thesis. The corecursive call is valid, because it is guarded by the `BarbSim_tau` observation and the proof covers all cases, because we analyzed all the observable properties that characterize strong barbed similarity.

3. This follows immediately from Lemma 2.1.4, which was also crucial for one implication of the Harmony Lemma.

□

```
rec res_barb_sim: (g:ctx) BarbSim [g,x:names ⊢ P] [g,x:names ⊢ Q] →
                  BarbSim [g ⊢ p_res \x.P] [g ⊢ p_res \x.Q] =
fun d .BarbSim_barb_in b ⇒
       PCtx_preserves_barb_in (mlam Y ⇒ fn b' ⇒ d .BarbSim_barb_in b') (presM pidM) b
  | d .BarbSim_barb_out b ⇒
       PCtx_preserves_barb_out (mlam Y ⇒ fn b' ⇒ d .BarbSim_barb_out b') (presM pidM) b
  | d .BarbSim_tau s ⇒ let [_ ⊢ fs_res \x.S] = s in
       let Ex_sim_barb_def [_,x:names ⊢ S'] b' = d .BarbSim_tau [_,x:names ⊢ S] in
       Ex_sim_barb_def [_ ⊢ fs_res \x.S'] (res_barb_sim b');
```

Figure 7: Encoding of the proof of Lemma 3.3.2.

We now recall the notion of *strong barbed simulation up to* $\dot{\leq}_B$, namely a binary relation on processes $\mathscr{R}$ such that:

1. $\mathscr{R}$ is barb preserving.
2. if $P \mathscr{R} Q$ and $P \xrightarrow{\tau} P'$, then there is Q' such that $Q \xrightarrow{\tau} Q'$ and $P' \dot{\leq}_B \mathscr{R} \dot{\leq}_B Q'$.

As above, *strong barbed similarity up to* $\dot{\leq}_B$ is the union of all strong barbed simulations up to $\dot{\leq}_B$.

Again, we cannot develop a general theory of relations up to: we simply encode strong barbed similarity up to $\dot{\leq}_B$ as we did for strong barbed similarity — the only difference being in the inductive auxiliary type, which encodes the *up to* $\dot{\leq}_B$.

In this setup, strong barbed similarity up to $\dot{\leq}_B$ coincides with $\dot{\leq}_B$. The use here of *up-to technique* is just for convenience: in order to prove that a relation is included in strong barbed similarity, it is sufficient to prove that it is a strong barbed simulation up to $\dot{\leq}_B$ and therefore included in barbed similarity up to $\dot{\leq}_B$.

3.1 Barbed precongruence

One way to distinguish two processes is to observe whether they exhibit the same behaviour when placed in the same environment, or context; a desirable property of a (bi)similarity is to be preserved by every possible environment. Unfortunately, strong barbed similarity does not satisfy contextual closure. For instance, the two processes $\bar{x}y.\bar{a}b.0$ and $\bar{x}y.0$ are strong barbed similar, since their only barb is $\downarrow_{\bar{x}}$ and they cannot perform any internal transition. Conversely, plugging them into the context $[\_] \mid x(z).0$ modifies their observable behavior: the former can perform a transition $\bar{x}y.\bar{a}b.0 \mid x(z).0 \xrightarrow{\tau} \bar{a}b.0 \mid 0$, enabling the observation of a barb $\downarrow_{\bar{a}}$ that the process $\bar{x}y.0 \mid x(z).0$, after an internal transition, cannot produce.

Therefore, we aim to identify those pairs of processes whose behavior remains strongly barbed similar under all contexts. The processes P and Q are *strongly barbed precongruent*, denoted by $P \leq_B Q$, if, for each context C, $C[P] \dot{\leq}_B C[Q]$. 

Among the properties of strong barbed precongruence, the most prominent is its characterization as the largest precongruence included in strong barbed similarity. It turns out that the latter can be formalized as a coinductive definition with only two observations, requiring the inclusion in $\dot{\leq}_B$ and the contextual closure. 

Lemma 3.4 (Properties of strong barbed precongruence)**.** *Strong barbed precongruence*

1. *is a preorder;* 
2. *is included in strong barbed similarity;* 
3. *is a precongruence;* 
4. *is the largest precongruence included in strong barbed similarity;* 
5. *includes structural congruence.* 

The first three proofs immediately follow from the definition of $\leq_B$, the fact that $\dot{\leq}_B$ is a preorder and context composition; the last two are straightforward coinductive arguments.

4 Context Lemma

The objective of CCFB3 is to prove that making barbed bisimilarity sensitive to substitutions and parallel composition is sufficient to establish barbed congruence. This result is an instance of a *context* lemma, namely a characterization of congruence that relaxes the universal quantification on arbitrary contexts, thereby simplifying proofs of congruence.

We recall that *substitutions* are endofunctions on names with finite support; they can be extended to processes and actions by simultaneously replacing all their free occurrences of names. To encode them, we rely on Beluga's built-in notion of simultaneous substitutions. In particular, given two Beluga contexts `g,h:ctx`, a substitution `$S:$[h |- g]` maps the names in `g` to names in `h`; given a process `P:[g |- proc]` that depends on names in `g`, the process `P[$S]:[h |- proc]` is obtained by replacing its names according to `$S`.

Next, we denote as $\leq_{|\sigma} := \{(P,Q) \mid \text{for all } R, \sigma, (P\sigma \mid R) \dot{\leq}_B (Q\sigma \mid R)\}$ the relation obtained by closing barbed similarity under parallel composition and substitutions. Analogously to barbed precongruence, it is formalized by an inductive type `BarbPre'`. 

The strategy to prove the context lemma consists in showing that $\leq_{|\sigma}$ is a precongruence included in strong barbed similarity; since, by Lemma 3.4.4, strong barbed precongruence is the largest precongruence included in strong barbed similarity, it follows that $\leq_{|\sigma} \subseteq \leq_B$. Below, we detail the results required to complete the proof.

First, an auxiliary lemma describing how transitions of a process $P\sigma$ arise from transitions of P:

Lemma 4.1. *If $P\sigma \xrightarrow{\alpha} P'$ and $\alpha \neq \tau$, then there are β, P'' such that $P \xrightarrow{\beta} P''$ and $\alpha = \beta\sigma$.* 

Proof. (Sketch) By inversion on α and then structural induction on P. In the Beluga encoding, the three cases $\alpha = x(y), \bar{x}y$ and $\bar{x}(y)$ are addressed separately. □

Proving that $\leq_{|\sigma} \subseteq \dot{\leq}_B$ is straightforward:

Lemma 4.2. *$\leq_{|\sigma}$ is included in strong barbed similarity.*

Proof. It is sufficient to observe that, if $P \leq_{|\sigma} Q$, then $P \equiv P(\mathsf{Id}) \mid 0 \;\dot{\leq}_B\; Q(\mathsf{Id}) \mid 0 \equiv Q$, where Id denotes the identity substitution. □

The proof that $\leq_{|\sigma}$ is a precongruence is trickier and is based on the following result — this is, in fact, the only instance in which coinductive up-to techniques are used.

Lemma 4.3. *The following relations are included in strong barbed similarity:*

1. $\mathscr{R}_1 \coloneqq \{(x(y).(P\sigma) \mid R,\ x(y).(Q\sigma) \mid R) \colon P \leq_{|\sigma} Q\}$.
2. $\mathscr{R}_2 \coloneqq \{(\bar{x}y.(P\sigma) \mid R,\ \bar{x}y.(Q\sigma) \mid R) \colon P \leq_{|\sigma} Q\}$.
3. $\mathscr{R}_3 \coloneqq \{(!(P\sigma) \mid R,\ !(Q\sigma) \mid R) \colon P \leq_{|\sigma} Q\}$.

Proof. We prove that $\mathscr{R}_1 \cup \dot{\leq}_B$ and $\mathscr{R}_2 \cup \dot{\leq}_B$ are strong barbed simulations, while $\mathscr{R}_3$ is a strong barbed simulation up to $\dot{\leq}_B$. The proofs, both as pen-and-paper and in Beluga, are carried out by coinduction, considering each possible observation and constructing the corresponding object required by the definition.

1. Checking that the elements of $\dot{\leq}_B$ satisfy all the observations is immediate, hence we focus on the elements of $\mathscr{R}_1$. Proving that $\mathscr{R}_1 \cup \dot{\leq}_B$ is barb preserving is also immediate, by looking at the structure of the pairs of processes in $\mathscr{R}_1$. To show that it is a reduction simulation, we proceed by inversion on the internal transition s that needs to be matched; in particular, we illustrate the case in which s has been derived from the S-COM-R rule, since it highlights one of the properties of substitutions that Beluga directly supports.

   The transition s has the form $x(z).P\sigma \mid R \xrightarrow{\tau} (P\sigma)\{y/z\} \mid R'$ for some y, R'; similarly, we can see that $x(z).Q\sigma \mid R \xrightarrow{\tau} (Q\sigma)\{y/z\} \mid R'$. As the composition of substitutions is a substitution, we conclude by observing that $((P\sigma)\{y/z\} \mid R',\ (Q\sigma)\{y/z\} \mid R') = (P(\sigma \circ \{y/z\}) \mid R',\ Q(\sigma \circ \{y/z\}) \mid R') \in \mathscr{R}_1$.
2. Analogous to the previous case.
3. Proving that $\mathscr{R}_3$ is barb preserving requires showing that a transition $s : !(P\sigma) \xrightarrow{\alpha} P'$, with $\alpha \neq \tau$, is matched by $!(Q\sigma)$. After inversion on s through the S-REP rule, the desired transition is built after applying Lemma 4.1.

   Conversely, proving the reduction closure up to $\dot{\leq}_B$ of $\mathscr{R}_3$ does not require any explicit inversion; rather, it is enough to apply Lemma 2.1.2 and the coinductive hypothesis to build a long chain of structural congruences and strong barbed similarities.

□

Lemma 4.4. *$\leq_{|\sigma}$ is a precongruence.*

Proof. By induction on the structure of the given context. The prefixes and replication cases make use of Lemma 4.3, while the others rely on chains of $\equiv$ and $\dot{\leq}_B$. □

Lemma 4.5. *$\leq_{|\sigma}$ is included in the largest precongruence included in strong barbed similarity.*

Proof. Follows immediately from Lemmas 4.2 and 4.4. □

Theorem 4.1 (Context Lemma)**.** $\leq_{|\sigma}$ *is included in strong barbed precongruence.*

Proof. Follows from Lemma 3.4.4, characterizing barbed precongruence as the largest precongruence included in strong barbed similarity. □

5 Evaluation

The entire development comprises approximately 1500 lines of code, organized into nine files. It includes 23 definitions — three of which are coinductive — accounting for roughly 10% of the total codebase, as well as 53 theorems. Most of these theorems exactly correspond to the relevant results presented in [18]; however, several of the more intricate ones must be decomposed into multiple statements. This is particularly necessary when they involve combinations of induction and coinduction with differing proof goals. Notably, the only strengthening lemmas required in the Beluga formalization are those already needed at the paper level to satisfy the side conditions of the LTS rules concerning free and bound names. This once again underscores the advantages of using HOAS for handling binders.

A central feature of the π-calculus is its ability to pass names, enabling subprocesses to communicate channel names and subsequently use them. Concretely, input prefixes bind received names, and in all standard semantics there is a transition in which the binder is eliminated and the bound name is replaced by a fresh free name representing the received value. In Beluga, such single substitutions are treated as a special case of simultaneous substitutions, which allows them to be composed seamlessly, as demonstrated in Lemma 4.3. This is a net gain compared to other approaches (such as [3]), where the two notions need to be implemented from scratch.

One complication arising from substitutions in Beluga is their interaction with the coverage checker. Verifying coverage often requires solving non-trivial unification problems, which can become more challenging in the presence of substitutions: this is well understood in the theory, but simply not supported yet by the Beluga implementation. Consequently, Lemma 4.1 is formalized without a totality check. In fairness, we find more urgent the implementation of the *productivity* checker: for what it is worth, we have manually checked that all coinductive calls comply with productivity, as we understand it, but this falls somewhat short of a complete assurance of the correctness of our mechanization.

In the CCFB paper, the authors stated:

> *The crux of our challenge is the effective use of coinductive up-to techniques. The intention is that the result should be relatively easy to achieve once the main properties of bisimilarity are established.*

However, coinductive up-to techniques, which have a prominent role in the study of other bisimilarities, are only used here once, i.e. to prove the context lemma, and the formalization of strong barbed similarity up $\dot{\leq}_B$ to does not diverge significantly from the formalization of strong barbed similarity. We conclude that the design of the challenge does not exercise significantly the up-to aspect.

6 Related work

In this section we mostly concentrate on mechanizations of behavioral equivalence in process calculi.

In [3, 2], Bengtson and Parrow provide the largest formalization of the π-calculus in the literature using nominal logic in Isabelle/HOL to represent binders. The encoding of the operational semantics uses *commitments* [14] rather than labelled transitions, i.e. pairs of an action and a derivative process that binds the bound names of the action in the derivative. The treatment of replication follows the single-rule presentation of [8]. Their results include proving that strong and weak bisimulations, both early and late, are congruences, and that structural congruence is a bisimulation; the authors also provide an axiomatization of strong late bisimulation and prove that it is sound and complete. Coinductive definitions and proofs are encoded by using Isabelle/HOL's standard coinductive package. Since most of these details are handled by Isabelle, the main focus of Bengtson and Parrow is on the nominal logic aspects of the formalization, which, while elegantly supporting free and bound names, still entails a fair amount of work to establish basic infrastructural properties.

Ambal et al. [1] provide a Coq formalization of the higher-order π-calculus that systematically studies binder representations for mixed binding (process vs name binders) and establishes strong context bisimilarity as a congruence via Howe's method. Coinduction is used in the standard way to characterize bisimilarity; coinductive "guarded" proofs only appear in the final step showing that the Howe closure is included in bisimilarity. A direct Coq development in the style of ours would likely require libraries such as PACO [10]. Further, the HOAS approach encodes process and name binders in the same way, offering significant simplifications.

In [20], Tian and Sangiorgi study proof methods for bisimulation in CCS, focusing on contractions rather than equations. They formalize the congruence induced by weak bisimilarity (rooted bisimilarity) in HOL4 using the coinductive package `Hol_coreln`. Their development does not treat processes up to α-conversion, and restriction is not a binder; contexts are therefore represented as single-binder λ-expressions applied to processes. As in our work, the coarsest congruence contained in bisimilarity is characterized via closure under composition, although here it is ancillary to the main result on unique solutions of rooted contractions.

Tiu and Miller [21] give a proof-search specification of the finite π-calculus in a logic equipped with definitions, fixed points, and the ∇ quantifier for generic judgments. Their encoding is equivalent to ours w.r.t. syntax and LTS. It does seem more general insofar that the interaction between the $\forall$, $\exists$, and ∇ quantifiers explains the usual distinctions between early, late, and, remarkably, open bisimulation. For the latter the ∇ quantifier plays a role analogous to Sangiorgi's *distinctions*.

Veltri and Vezzosi [22] give a much more extensive formalization of process-calculus semantics in Guarded Cubical Agda: they develop fully abstract denotational models for CCS and the early π-calculus, showing that denotational equality coincides with bisimilarity. Their encoding uses well-scoped de Bruijn syntax, in contrast with our HOAS representation in Beluga. Their approach to coinduction is also quite different: guarded recursion, clocks, and cubical path equality provide the semantic infrastructure through which bisimilarity is recovered as equality, whereas we reason directly with coinductive operational relations.

7 Conclusions and Future Work

What happens if a challenge turns out to be not too challenging? Is it the designers' fault or the encoders' merit? In our previous paper we mused on how uneventful the formalization had been. We have to repeat ourselves: the HOAS encoding of the syntax and semantics of the calculus, which we have inherited from a long tradition (dating back to [13]), abstracts all the issues related to free and bound names that have preoccupied all first-order encodings. What the present paper brings to the table is a very intensive

use of coinductive reasoning: here, Beluga's copattern approach shines, much more than in a previous coinductive case study (Howe's method, see [15], which turns out to exercise coinduction only in a limited fashion).

Beluga's underlying sized types discipline is intrinsically compositional, making nesting coinductive proofs completely unproblematic. This is in sharp contrast with the guarded recursion approach native to Rocq, which would have been simply unfeasible and compelled the user to switch to dedicated libraries such as PACO [10]. Again, it is the support for both HOAS and coinduction via observation that makes, in our judgment, the development so smooth — Agda supports a similar approach to coinduction, but not HOAS; Abella offers the converse trade-off.

Future work. There are several directions in which the present formalization could be extended. A natural first step is to establish the converse direction of the context lemma. The proof presented in [18] proceeds by constructing a suitable context such that a pair of strongly barbed congruent processes can be composed in parallel under substitution so as to yield strongly barbed bisimilar processes. The argument relies on multi-step internal transitions whose length depends on the size of the support of the substitution. It is not clear to us whether Beluga's theory of substitution directly captures this style of argument, particularly when it involves a fine-grained analysis of substitutions and their supports. Moreover, the proof makes essential use of the symmetry of strong barbed bisimilarity, so we would have to carry out the symmetric proofs after all.

Open bisimulation is particularly appealing in our setting because Beluga's contextual objects and simultaneous substitutions seem to provide much of the infrastructure needed to express substitution-closed behavioural relations. The main additional burden would be the treatment of distinctions/freshness constraints and their interaction with the LTS. Since Beluga does not have a ∇ quantifier, one must prove that the resulting substitution-based relation coincides with the standard distinction-indexed definition, or else identify precisely which variant of open similarity has been mechanized. One possibility to avoid an explicit representation of distinctions could be to look at the theory of *quasi-open* bisimilarity [6, 9].

Another technically involved extension concerns the coincidence between strong barbed congruence and early congruence. The standard proof requires the introduction of a stratification of strong early bisimilarity, with the key inclusion shown via a contrapositive argument. This reliance on classical reasoning renders a direct mechanization in Beluga particularly challenging.

Finally, Lassen in [12] studies contextual closure as the greatest *adequate* congruence in the context of program equivalence for a functional language. Since adequacy has a barbed "flavour", the question is whether analogous characterizations can be established for bisimilarity-based congruences in the π-calculus.

Acknowledgments

This work is partially supported by the National Science Foundation under Grant No. 2242786 (SHF:Small:Concurrency In Reversible Computations). We thank Brigitte Pientka for several helpful conversations.

AI usage statement. The formalization, definitions, and proofs presented in this paper are entirely the authors' work. Generative AI tools were used solely for editorial assistance (improving clarity and English wording), and had no role in the technical development or validation of the results.

References

[1] Guillaume Ambal, Sergueï Lenglet & Alan Schmitt (2021): *HOπ in Coq*. J. Autom. Reason. 65(1), pp. 75–124, doi:10.1007/S10817-020-09553-0.

[2] Jesper Bengtson & Joachim Parrow (2007): *A Completeness Proof for Bisimulation in the π-calculus Using Isabelle*. Electronic Notes in Theoretical Computer Science 192, pp. 61–75, doi:10.1016/J.ENTCS.2007.08.017.

[3] Jesper Bengtson & Joachim Parrow (2009): *Formalising the π-Calculus using Nominal Logic*. Log. Methods Comput. Sci. 5, doi:10.2168/LMCS-5(2:16)2009.

[4] Marco Carbone, David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, Frederik Krogsdal Jacobsen, Alberto Momigliano, Luca Padovani, Alceste Scalas, Dawit Tirore, Martin Vassor, Nobuko Yoshida & Daniel Zackon (2024): *The Concurrent Calculi Formalisation Benchmark*. In Ilaria Castellani & Francesco Tiezzi, editors: Coordination Models and Languages, Springer Nature Switzerland, Cham, pp. 149–158, doi:10.1007/978-3-031-62697-5_9.

[5] Gabriele Cecilia & Alberto Momigliano (2024): *A Beluga Formalization of the Harmony Lemma in the π-Calculus*. In Florian Rabe & Claudio Sacerdoti Coen, editors: Proceedings Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, LFMTP@FSCD 2024, Tallinn, Estonia, 8th July 2024, EPTCS, pp. 1–17, doi:10.4204/EPTCS.404.1.

[6] Yuxi Fu (2005): *On quasi-open bisimulation*. Theor. Comput. Sci. 338(1-3), pp. 96–126, doi:10.1016/J.TCS.2004.10.041.

[7] Simon J. Gay & Vasco T. Vasconcelos (2025): *Session Types*. Cambridge University Press, doi:10.1017/9781009000062.

[8] Furio Honsell, Marino Miculan & Ivan Scagnetto (2001): *π-Calculus in (Co)Inductive Type Theory*. Theor. Comput. Sci. 253(2), pp. 239–285, doi:10.1016/S0304-3975(00)00095-5.

[9] Ross Horne, Ki Yung Ahn, Shang-Wei Lin & Alwen Tiu (2018): *Quasi-Open Bisimilarity with Mismatch is Intuitionistic*. In Anuj Dawar & Erich Grädel, editors: Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018, ACM, pp. 26–35, doi:10.1145/3209108.3209125.

[10] Chung-Kil Hur, Georg Neis, Derek Dreyer & Viktor Vafeiadis (2013): *The Power of Parameterization in Coinductive Proof*. In: POPL '13: Proc. 40th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, ACM, New York, p. 193–206, doi:10.1145/2429069.2429093.

[11] Adrienne Lancelot, Beniamino Accattoli & Maxime Vemclefs (2025): *Barendregt's Theory of the λ-Calculus, Refreshed and Formalized*. In Yannick Forster 0001 & Chantal Keller, editors: 16th International Conference on Interactive Theorem Proving, ITP 2025, September 28 to October 1, 2025, Reykjavik, Iceland, LIPIcs 352, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, doi:10.4230/LIPIcs.ITP.2025.13.

[12] Søren Bøgh Lassen (1998): *Relational Reasoning about Functions and Nondeterminism*. Doctoral dissertation, Faculty of Science of the University of Aarhus. Available at `https://www.brics.dk/DS/98/2/`.

[13] Dale Miller (1994): *Specification of the π-Calculus*. Available at `http://www.lix.polytechnique.fr/Labo/Dale.Miller/lProlog/examples/pi-calculus/toc.html`.

[14] Robin Milner (1993): *The Polyadic π-Calculus: a Tutorial*. In Friedrich L. Bauer, Wilfried Brauer & Helmut Schwichtenberg, editors: Logic and Algebra of Specification, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 203–246, doi:10.1007/978-3-642-58041-3_6.

[15] Alberto Momigliano, Brigitte Pientka & David Thibodeau (2019): *A Case-Study in Programming Coinductive Proofs: Howe's Method*. Math. Struct. Comput. Sci. 29(8), pp. 1309–1343, doi:10.1017/S0960129518000415.

[16] Hiroshi Nakano (2000): *A Modality for Recursion*. In: 15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000, IEEE Computer Society, pp. 255–266, doi:10.1109/LICS.2000.855774.

[17] Damien Pous & Dmitriy Traytel (2026): *Handbook of Proof Assistants*, chapter Coinductive methods. Springer.

[18] Davide Sangiorgi & David Walker (2001): *The π-Calculus - a Theory of Mobile Processes*. Cambridge University Press, doi:10.2178/bsl/1182353926.

[19] David Thibodeau, Andrew Cave & Brigitte Pientka (2016): *Indexed Codata Types*. In Jacques Garrigue, Gabriele Keller & Eijiro Sumii, editors: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, ACM, pp. 351–363, doi:10.1145/2951913.2951929.

[20] Chun Tian & Davide Sangiorgi (2020): *Unique Solutions of Contractions, CCS, and their HOL Formalisation*. *Inf. Comput.* 275, p. 104606, doi:10.1016/J.IC.2020.104606.

[21] Alwen Tiu & Dale Miller (2010): *Proof Search Specifications of Bisimulation and Modal Logics for the π-Calculus*. *ACM Trans. Comput. Log.* 11(2), pp. 13:1–13:35, doi:10.1145/1656242.1656248.

[22] Niccolò Veltri & Andrea Vezzosi (2023): *Formalizing CCS and π-calculus in Guarded Cubical Agda*. *J. Log. Algebraic Methods Program.* 131, p. 100846, doi:10.1016/J.JLAMP.2022.100846.

[23] Daniel Zackon, Chuta Sano, Alberto Momigliano & Brigitte Pientka (2025): *Split Decisions: Explicit Contexts for Substructural Languages*. In Kathrin Stark, Amin Timany, Sandrine Blazy & Nicolas Tabareau, editors: *Proceedings of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2025, Denver, CO, USA, January 20-21, 2025*, ACM, pp. 257–271, doi:10.1145/3703595.3705888.