HAL open science

HOCore in Coq

Martín Escarrá, Petar Maksimović, Alan Schmitt

To cite this version:

Martín Escarrá, Petar Maksimović, Alan Schmitt. HOCore in Coq. Vingt-sixièmes Journées Francophones des Langages Applicatifs (JFLA 2015), Jan 2015, Le Val d'Ajol, France. hal-01099130

HAL Id: hal-01099130

https://inria.hal.science/hal-01099130

Submitted on 31 Dec 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HOCore in Coq

Martín Escarrá[2] & Petar Maksimović[1,3] & Alan Schmitt[1]

1: Inria
2: Universidad Nacional de Rosario
3: Mathematical Institute of the Serbian Academy of Sciences and Arts

Abstract

We consider a recent publication on higher-order process calculi [13] and describe how its main results have been formalized in the Coq proof assistant. We highlight several important technical issues that we have uncovered in the original publication. We believe these issues are not unique to the paper under consideration and require particular care to be avoided. Our ultimate goal is to show that it is possible to build a solid, high-confidence setting for formal reasoning on higher-order process calculi.

1. Introduction

Computer-aided verification has reached a point where it can be applied to state-of-the-art research domains, including compilers, programming languages, and mathematical theorems. Our goal is to contribute to this growing effort by formalizing a recent paper on process calculi.

Our main motivation is the increasing complexity of proofs in this domain. Simply put, researchers are forced to pay very close attention to a great number of details, some of which may remain implicit or go unnoticed, leaving room for imprecision or errors in the proofs. To overcome such issues, we argue a more formal treatment is needed. With our formalization, we strive for a deeper understanding of the proving process and a precise exposition of the overlooked details. Moreover, we show that it is possible to build a solid, high-confidence setting for formal reasoning on higher-order process calculi.

The paper that we are examining is [13], by Lanese et al. It introduces HOCore—a minimal higher-order process calculus, which features input-prefixed processes, output processes, and parallel composition. Its syntax is as follows:

$$P ::= a(x).P \;\Big|\; \overline{a}\langle P\rangle \;\Big|\; P \parallel P \;\Big|\; x \;\Big|\; \mathbf{0}.$$

HOCore is minimal, in the sense that it features only the operators strictly necessary for higher-order communication. For one, there are no continuations following output messages. More importantly, there is no restriction operator, rendering all channels global and the dynamic creation of new channels impossible. The semantics of HOCore is presented in [13] in the form of a labeled transition system and it is shown that HOCore is a Turing-complete calculus. On the other hand, its observational equivalence is proven to be decidable. In fact, the main forms of strong bisimilarity considered in the literature (contextual equivalence, barbed congruence, higher-order bisimilarity, context bisimilarity, and normal bisimilarity [5, 12, 21, 25]) all coincide in HOCore. Moreover, their synchronous and asynchronous versions coincide as well. It is thus possible to decide that two processes are equivalent, yet it is impossible, in the general case, to decide whether or not they terminate. In addition, the authors give in [13] a sound and complete axiomatization of observational equivalence, and show that its decidability breaks in the presence of four static restrictions. Our

formalization focuses on the subset of [13] addressing the decidability of observational equivalence and the coincidence of synchronous bisimilarities, but also includes an initial effort towards the formalization of the soundness and completeness of the axiomatization of HOCore.

Our theorem prover of choice for this development is Coq [15]. Coq allows the user to describe concepts such as mathematical theories or programming languages, state properties related to these concepts, and verify that such properties hold. Proofs produced using Coq—and interactive theorem provers in general—have a higher degree of reliability when compared to pen-and-paper proofs. They rely on a *trusted core*—the code on which the theorem prover is based—upon which all subsequent inferences are made. Coq supports inductive and co-inductive reasoning, offers a variety of tactics to build proofs, provides a comprehensive library, and permits extraction of a certified program from the constructive proof of its formal specification. Its typical applications include the formalization of programming languages semantics (e.g., the CompCert compiler certification project [14]), the formalization of mathematics (e.g., the formal proof of the four color theorem [8]), and teaching [18].

Contributions. We present a formalization of HOCore, both the calculus itself and its behavioral theory. We first introduce the syntactic and semantic concepts of HOCore: processes, freshness of variables and channels, structural congruence, and the labeled transition system (Section 2). We focus in particular on how to model bound variables using the canonical approach of Pollack et al. [19]. We then turn to the formalization of the various strong bisimilarities used in [13]: higher-order, context, normal, open normal, and IO-bisimilarity (Section 3).

The main contributions of the paper are the formal proofs that these five bisimilarities coincide and that they are decidable. We detail the errors, inaccuracies, and implicit assumptions made in the pen-and-paper proofs, and show how to correct them (Section 4). These issues are not unique to the paper under consideration, and require particular care to be avoided.

Finally, we strive to execute the formalization so as to preserve a high degree of both visual and technical correspondence between the formulations of theorems and proofs on paper and their counterparts in Coq. We also gained valuable insights into the interplay between the flexibility of process calculi and the rigidity of a formal verification environment. We summarize what we have learned in the process in Section 6.

The formalization of the calculus, in particular our choice of binding technology, and the decidability of IO bisimilarity have been presented in [3]. They are recalled here so that the paper be self-contained. This paper extends that work to cover all the results of [13] regarding bisimulations.

The complete formalization, available online,[1] consists of approximately 4 thousands of lines of code (kloc) of specification and 13 kloc of proofs. It was developed intermittently over a period of three years by the three authors and Simon Boulier. In the rest of the paper, we use the symbols as links to relevant definitions or lemmas in the online Coq code. Our Coq development heavily relies on the TLC library, developed by Arthur Charguéraud.[2] TLC is a general purpose Coq library that offers many tactics for proof automation.

2. Formalizing HOCore

2.1. Syntax

HOCore is a minimal higher-order process calculus, based on the core of well-known calculi such as CHOCS [24], Plain CHOCS [26], and Higher-Order π-calculus [21, 22, 23]. The main syntactic categories of HOCore are channels (a, b, c), variables (x, y, z), and processes (P, Q, R). Channels and

[1] http://www.irisa.fr/celtique/aschmitt/research/hocore/
[2] http://www.chargueraud.org/softs/tlc/

variables are atomic, whereas processes are defined inductively.

$$
\begin{array}{rll}
P,\ Q ::= & \overline{a}\langle P\rangle & \text{output process} \\
& \mid a(x).P & \text{input prefixed process} \\
& \mid x & \text{process variable} \\
& \mid P \parallel Q & \text{parallel composition} \\
& \mid \mathbf{0} & \text{empty process}
\end{array}
$$

An input-prefixed process $a(x).P$ awaits to receive a process, for example Q, on some channel a; it then replaces every occurrence of the variable x in the process P with Q (similarly to a λ-abstraction). An output process[3] $\overline{a}\langle Q\rangle$ emits the process Q on channel a. Parallel composition enables the interaction between the senders and the receivers.

The only binding in HOCore occurs for variables in input processes. We denote the free variables of a process P by $\mathsf{fv}(P)$, and the bound ones by $\mathsf{bv}(P)$. In [13], processes are identified up to the renaming of bound variables. Processes that do not contain free variables are *closed* whereas those that do are *open*. If a variable x does not occur either free or bound within a process P, we say that it is fresh with respect to that process, and denote this in the Coq development by $x \sharp P$. We denote lists with the symbol $\widetilde{\ }$: a list of variables is, for instance, denoted by $\widetilde{x}$.

Structural congruence. The structural congruence relation, written $\equiv$, is the smallest congruence relation on processes for which parallel composition is associative, commutative, and has $\mathbf{0}$ as the neutral element. In our formalization, we distinguish a single step of structural congruence from its reflexive-transitive closure, denoted by $\equiv^*$.

$$
P \parallel Q \equiv Q \parallel P \qquad P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \qquad P \parallel \mathbf{0} \equiv P
$$

Canonical representation of terms. The first step of our formalization addresses the treatment of bound variables and α-conversion. We have opted for the *locally named* approach, as described by Pollack et al. in [19]. In this approach, the set of variables is explicitly divided into two distinct subsets: free (global) variables, denoted by X, Y, Z, and bound (local) variables, denoted by x, y, z. The main idea is to dispense with α-equivalence altogether by calculating the value of a local variable canonically at the time of its binding. For this calculation, we rely on a *height function* f, taking as arguments the global variable X and the term P within which it is to be bound, and computing the value of the corresponding local variable x. We then replace every occurrence of X with x in P.

As an illustration, let us examine the process $a(x).(x \parallel x)$. This process is α-equivalent to the process $a(y).(y \parallel y)$, for any local variable y, and we would like to be able to choose a canonical representative for this entire family of processes. To that end, we start from the process $X \parallel X$, where the variable X is free, calculate $x = f_X(X \parallel X)$, and take this x to be our canonical representative. In the following, we write $a[X]_f.P$ for $a(f_X(P)).[f_X(P)/X]P$.

The height function needs to meet several criteria so that it does not, for one, return a local variable already used in a binding position around the global variable to be replaced, thus resulting in a capture. We use the three criteria required by the authors in [19] that guarantee that the height function is well-defined. ( ) ( ) ( ) Although it is not strictly necessary to choose a particular height function f that satisfies these criteria, it is important to demonstrate that such a function exists, which we do, using the following function and representing local variables as integers. ( ) ( )

$$
\begin{array}{lll}
f_X(X) = 1 & f_X(y) = 0 & f_X(\mathbf{0}) = 0 \\
f_X(Y) = 0 \quad \text{if } X \neq Y & f_X(\overline{a}\langle P\rangle) = f_X(P) & f_X(P \parallel Q) = \mathtt{max}(f_X(P), f_X(Q))
\end{array}
$$

[3] We will also refer to an output process as an output message or, more simply, a message.

$$f_X(a(y).P) = \begin{cases} f_X(P) & \text{if } f_X(P) = 0 \text{ or } f_X(P) > y \\ y+1 & \text{otherwise} \end{cases}$$

Adjusting the syntax. The separation of variables into local and global ones is reflected in the syntax of processes

$$P,\ Q ::= a(x).P \ \mid\ \overline{a}\langle P\rangle \ \mid\ P \parallel Q \ \mid\ x \ \mid\ X \ \mid\ \mathbf{0}$$

which is encoded in Coq inductively as follows:

```
Inductive process : Set :=
 | Send    : chan    -> process -> process
 | Receive : chan    -> lvar    -> process -> process
 | Lvar    : lvar    -> process
 | Gvar    : var     -> process
 | Par     : process -> process -> process
 | Nil     : process.
```

In this definition, `chan`, `lvar`, and `var` are types representing channels, local variables, and global variables, respectively.[4] We can then define the substitution operation without any complications arising from the renaming of variables. We write $[Q/X]P$ for the substitution of the variable X with the process Q in P, and $[\widetilde{Q}/\widetilde{X}]P$ for the simultaneous substitution of distinct variables $\widetilde{X}$ with processes $\widetilde{Q}$ in P. In this paper, we use the same notation for both operations; the Coq version uses two distinct notations. ( ) ( )

$$\begin{aligned}
[P/X]X &= P \\
[P/X]Y &= Y \quad \text{if } X \neq Y \\
[P/X]x &= x \\
[P/X]\mathbf{0} &= \mathbf{0} \\
[P/X](Q \parallel R) &= [P/X]Q \parallel [P/X]R \\
[P/X]\overline{a}\langle Q\rangle &= \overline{a}\langle [P/X]Q\rangle \\
[P/X]a(y).Q &= a(y).[P/X]Q
\end{aligned}
\qquad
\begin{aligned}
[P/x]x &= P \\
[P/x]y &= y \quad \text{if } x \neq y \\
[P/x]X &= X \\
[P/x]\mathbf{0} &= \mathbf{0} \\
[P/x](Q \parallel R) &= [P/x]Q \parallel [P/x]R \\
[P/x]\overline{a}\langle Q\rangle &= \overline{a}\langle [P/x]Q\rangle \\
[P/x]a(y).Q &= a(y).[P/x]Q \quad \text{if } x \neq y \\
[P/x]a(x).Q &= a(x).Q
\end{aligned}$$

Well-formed processes. Given a height function f, we now define *well-formed processes* as processes in which all local variables are bound[5] and are computed using f. We refer to them as *wf-processes*. We thus define the predicate `wf` that inductively characterize well-formed processes. ( )

Since all of the subsequent definitions, such as those of the labeled transition system, congruence, and bisimulations, take into consideration wf-processes only, we define a dependent type corresponding to wf-processes, by pairing processes with the proof that they are well-formed.

```
Record wfprocess : Set := mkWfP {proc :> process; wf_cond : wf proc}.
```

A `wfprocess` is a record with one constructor, `mkWfP`. This record contains two fields: a process, named `proc`, and a proof that `proc` is well-formed, named `wf_cond`. This type is *dependent* because

[4] To be precise, `chan` and `lvar` are aliases for Coq's type for natural numbers (`nat`), while `var` is the type for global variables provided by the TLC library.

[5] Here we have a slight overloading of terminology when it comes to free and bound variables—while local variables are intended to model the bound variables of the object language, they can still appear free with respect to our adjusted syntax. For example, x is free in the process `Lvar x`, and we do not consider this process to be well-formed.

the type of the second field (the type of a proof that `proc` is well-formed, i.e., `wf proc`) depends on the value of the first field. The notation `proc :> process` is a *coercion*: it automatically transforms a `wfprocess` into a `process` through the `proc` field when needed.

This approach allows us to directly reason about wf-processes without having to carry additional hypotheses in lemmas or definitions. Each time a wf-process is constructed, however, we are required to provide a proof of its well-formedness. To this end, we introduce well-formed counterparts for all of the process constructors. For instance, to express parallel composition of wf-processes, we use the following definition:

```
Definition wfp_Par (p q : wfprocess) :=
  mkWfP (Par p q) (WfPar p q (wf_cond p) (wf_cond q)).
```

where the constructor `WfPar` is used to build the proof of well-formedness of parallel processes:

```
| WfPar  : forall (p q : process), wf p -> wf q -> wf (Par p q).
```

Our initial development did not use dependent types, but instead relied on additional well-formedness hypotheses for most lemmas. We found out that, in practice, it is much more convenient to bundle processes and well-formedness together. We can, for instance, use Coq's built-in notion of reflexive relations without having to worry about relating non well-formed processes. Also, we can define transitions and congruence directly on well-formed processes, thus streamlining the proofs by removing a significant number of lines of code otherwise required to show that processes obtained and constructed during the proofs are indeed well-formed.

2.2. Semantics

The original semantics of HOCore is expressed via a labeled transition system (LTS) applied to a calculus with α-conversion. It features the following transition types: internal transitions $P \xrightarrow{\tau} P'$, input transitions $P \xrightarrow{a(x)} P'$, where P receives on the channel a a process that is to replace a local variable x in the continuation P', and output transitions $P \xrightarrow{\overline{a}\langle P'' \rangle} P'$, where P emits the process P'' on channel a and evolves to P'. This LTS is not satisfactory because the input transition mentions the name of its bound variable. For instance, the following transition needs to be prevented to avoid name capture: $x \parallel a(x).x \xrightarrow{a(x)} x \parallel x$. To this end, side conditions are introduced and the entire semantics is defined "up-to α-conversion" to make sure terms do not get stuck because of the choice of a bound name. To obtain a simpler formalization, we restate it using abstractions.

Abstractions and agents. An *agent* A is either a process or an abstraction F. An *abstraction* can be viewed as a λ-abstraction inside a context.

$$A ::= P \;\mid\; F \qquad F ::= (x).P \;\mid\; F \parallel P \;\mid\; P \parallel F$$

We call these abstractions *localized*, because the binder (x) does not move (the usual approach to abstractions involves "lifting" the binder so that the abstraction always remains of the form $(x).P$). Our approach, similar to the one in [27], allows us to avoid the recalculation of bound variables.

The application of an abstraction to a process, denoted by $F \bullet Q$, follows the inductive definition of abstractions.

$$((x).P) \bullet Q = [Q/x]P \qquad (F \parallel P) \bullet Q = (F \bullet Q) \parallel P \qquad (P \parallel F) \bullet Q = P \parallel (F \bullet Q)$$

Removal transitions. To simplify the definition of bisimulations that allow an occurrence of a free variable to be deleted from related processes, we also add transitions $X \xrightarrow{X} \mathbf{0}$, where a global variable dissolves into an empty process.

The Labeled Transition System. The LTS consists of the following seven rules.

Inp $a(x).P \xrightarrow{a} (x).P$ **Out** $\overline{a}\langle P\rangle \xrightarrow{\overline{a}\langle P\rangle} \mathbf{0}$ **Rem** $X \xrightarrow{X} \mathbf{0}$

Act1 If $P \xrightarrow{\alpha} A$, then $P \parallel Q \xrightarrow{\alpha} A \parallel Q$. **Act2** If $Q \xrightarrow{\alpha} A$, then $P \parallel Q \xrightarrow{\alpha} P \parallel A$.

Tau1 If $P \xrightarrow{\overline{a}\langle P''\rangle} P'$ and $Q \xrightarrow{a} F$, then $P \parallel Q \xrightarrow{\tau} P' \parallel (F \bullet P'')$.

Tau2 If $P \xrightarrow{a} F$ and $Q \xrightarrow{\overline{a}\langle P''\rangle} Q'$, then $P \parallel Q \xrightarrow{\tau} (F \bullet P'') \parallel Q'$.

In these rules, α denotes an arbitrary transition label. In Coq, we represent transitions inductively, providing a constructor for each of the rules. ( ) To illustrate, we present the constructors that correspond to the rules Out and Rem.

```
| TrOut : forall a p, (transition (Send a p) (LabOut a p) (AP Nil))
| TrRem : forall X,   (transition (Gvar X) (LabRem X) (AP Nil))
```

For better legibility of the code, we introduce a special notation for transitions.

```
Notation "{ p -- l ->> ap }" := (transition p l ap) (at level 60).
```

3. Formalizing Bisimulations

We present several forms of strong bisimilarity commonly used in higher-order process calculi and describe their formalization in Coq. We first define a number of basic bisimulation clauses, as in [13].

Definition 1. *A relation $\mathcal{R}$ on HOCore processes is:*

- *a* variable *relation, if when $P \mathcal{R} Q$ and $P \xrightarrow{X} P'$, there exists a process Q', such that $Q \xrightarrow{X} Q'$ and $P' \mathcal{R} Q'$.* ( )
- *an* input *relation, if when $P \mathcal{R} Q$ and $P \xrightarrow{a} F$, there exists an abstraction F', such that $Q \xrightarrow{a} F'$, and for all global variables X fresh in P and Q, it holds that $(F \bullet X) \mathcal{R} (F' \bullet X)$.* ( )
- *an* input normal *relation, if when $P \mathcal{R} Q$ and $P \xrightarrow{a} F$, there exists an abstraction F', such that $Q \xrightarrow{a} F'$, and for all channels m fresh in P and Q it holds that $(F \bullet \overline{m}\langle\mathbf{0}\rangle) \mathcal{R} (F' \bullet \overline{m}\langle\mathbf{0}\rangle)$.* ( )
- *a τ-relation, if when $P \mathcal{R} Q$ and $P \xrightarrow{\tau} P'$, then there exists a process Q', such that $Q \xrightarrow{\tau} Q'$ and $P' \mathcal{R} Q'$.* ( )
- *an* output *relation, if when $P \mathcal{R} Q$ and $P \xrightarrow{\overline{a}\langle P''\rangle} P'$, then there exist processes Q' and Q'', such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$, $P' \mathcal{R} Q'$ and $P'' \mathcal{R} Q''$.* ( )
- *an* output normal *relation, if when $P \mathcal{R} Q$ and $P \xrightarrow{\overline{a}\langle P''\rangle} P'$, then there exist processes Q' and Q'', such that $Q \xrightarrow{\overline{a}\langle Q''\rangle} Q'$, and for all channels m fresh in P and Q and all global variables X fresh in P'' and Q'', it holds that $(m[X]_f.P'' \parallel P') \mathcal{R} (m[X]_f.Q'' \parallel Q')$.* ( )

- *an* output context *relation, if when* $P \mathcal{R} Q$ *and* $P \xrightarrow{\overline{a}\langle P'' \rangle} P'$*, then there exist processes* Q' *and* Q''*, such that* $Q \xrightarrow{\overline{a}\langle Q'' \rangle} Q'$*, and for all global variables* X *and processes* S*, such that* $\mathsf{fv}(S) \subseteq \{X\}$*, it holds that* $([P''/X]S \parallel P') \mathcal{R} ([Q''/X]S \parallel Q')$*.* ( )
- closed*, if when* $P \mathcal{R} Q$ *and* $P \xrightarrow{a} F$*, then there exist a process* Q' *and an abstraction* F'*, such that* $Q \xrightarrow{a} F'$*, and for all closed processes* R*, it holds that* $(F \bullet R) \mathcal{R} (F' \bullet R)$*.* ( )

```
Definition var_relation (R : RelWfP) : Prop :=
  forall (p q : wfprocess), (R p q) ->
    forall (X : var) (p' : wfprocess), {p -- LabRem X ->> (AP p')} ->
      exists (q' : wfprocess), {q -- LabRem X ->> (AP q')} /\ (R p' q').
```

There are two main differences between these definitions and the ones presented in [13]. The first one, as mentioned before, is the use of abstractions in place of variables in input transitions. The second difference is found in the definition of a *variable relation*; we use the transition **Rem**, while the one in [13] uses structural congruence:

- A symmetric relation $\mathcal{R}$ on HOCore processes is a variable bisimulation if when $P \mathcal{R} Q$ and $P \equiv x \parallel P'$, then there exists Q' such that $Q \equiv x \parallel Q'$ and $P' \mathcal{R} Q'$.

This particular change was motivated by the fact that we have chosen to define bisimulations semantically, in terms of processes that are capable of executing transitions. In that context, preserving the definition of a variable relation from [13] and involving structural congruence would be inconsistent. Moreover, working under the hypothesis that two processes are structurally congruent is very inconvenient, as their structure may be completely different.

Using these clauses, we define the five forms of strong bisimulation: higher-order ($\sim_{\mathtt{HO}}$), context ($\sim_{\mathtt{CON}}$), normal ($\sim_{\mathtt{NOR}}$), IO ($\sim^{\circ}_{\mathtt{IO}}$), and open normal ($\sim^{\circ}_{\mathtt{NOR}}$) bisimulations. Due to space restrictions, we present here only the ones that differ from those in [13].

Definition 2. *A relation* $\mathcal{R}$ *on HOCore processes is:*

- *an* IO*-bisimulation* ($\sim^{\circ}_{\mathtt{IO}}$) *if it is symmetric, an input relation, an output relation, and a variable relation;* ( )
- *an* open normal *bisimulation* ($\sim^{\circ}_{\mathtt{NOR}}$) *if it is symmetric, a* τ*-relation, an input relation, an output normal relation, and a variable relation.* ( )

```
Definition IO_bisimulation (R : RelWfP) : Prop :=
 (Symmetric R) /\ (in_relation R) /\ (out_relation R) /\ (var_relation R).
```

For each of these bisimulations, we also consider a corresponding *bisimilarity*, i.e., the union of all corresponding bisimulations, and each of these bisimilarities is proven to be a bisimulation itself. Bisimilarity can naturally be viewed as a co-inductive notion, but its co-inductive aspects can also be expressed in a set-theoretic way, as follows:

```
Definition IObis (p q : wfprocess) : Prop :=
  exists R, (IO_bisimulation R) /\ (R p q).
```

To compare the set-theoretic and the native Coq co-inductive versions of bisimilarity, we have also defined the latter ( ) and proven it equivalent to the set-theoretic one by using a variation of

Park's principle [7]. We can use these two definitions interchangeably, depending on which one can is easier to use in a given proof. For instance, the co-inductive definition is more convenient when proving statements in which the candidate relations are very simple and follow the formulation of the statement, as there is thus no need to supply the relation during the proof process. In the cases where the candidate relations are more complicated, however, we find it more natural and closer to the paper proofs to use the set-theoretic definition.

We also define the extensions of $\sim_{\texttt{HO}}$, $\sim_{\texttt{CON}}$, and $\sim_{\texttt{NOR}}$ to open processes by adding abstractions that bind all free variables, and denote these extensions by $\sim^{\star}_{\texttt{HO}}$, $\sim^{\star}_{\texttt{CON}}$, and $\sim^{\star}_{\texttt{NOR}}$, respectively. ( ) ( ) ( )

Definition 3. *Let $\mathcal{R}$ be a bisimilarity on closed HOCore processes. The extension of $\mathcal{R}$ to open HOCore processes, denoted by $\mathcal{R}^{\star}$, is defined by*

$$\mathcal{R}^{\star} = \{(P, Q) : a[x_1]_f. \ldots a[x_n]_f.P \ \mathcal{R} \ a[x_1]_f. \ldots a[x_n]_f.Q\}$$

where $fv(P) \cup fv(Q) = \{x_1 \ldots x_n\}$.

Relations and Bisimilarities "up-to". An important tool used in our development is the notion that two processes P and Q are in a relation $\mathcal{R}$ *up to* another relation $\mathcal{R}'$: assuming the processes are related by $\mathcal{R}$ before a transition, they are related by $\mathcal{R}'\mathcal{R}\mathcal{R}'$ after the transition.

More formally, we have the following (giving only one example definition in each category, the remaining ones being formulated analogously).

Definition 4. *A relation $\mathcal{R}$ on HOCore processes is:*

- *a τ-relation up to $\mathcal{R}'$, if when $P \ \mathcal{R} \ Q$ and $P \xrightarrow{\tau} P'$, there exists a process Q', such that $Q \xrightarrow{\tau} Q'$ and $P' \ \mathcal{R}'\mathcal{R}\mathcal{R}' \ Q'$.* ( )
- *an* IO*-bisimulation up to $\mathcal{R}'$ if it is symmetric, and if it is an input relation, an output relation, and a variable relation up to $\mathcal{R}'$.* ( )

Further, we establish the following connection between bisimilarities and bisimilarities "up-to", illustrated here only for open normal bisimilarity, but holding for all five.

Lemma 1. Let $\mathcal{R}'$ be an equivalence relation that is also an open normal bisimulation. Then, $p \sim^{\circ}_{\texttt{NOR}} q$ if and only if $p \sim^{\circ}_{\texttt{NOR}} q$ up to $\mathcal{R}'$. ( )

We mainly use this technique to reason up to structural congruence. We prove that structural congruence is an equivalence relation ( ), and that it is, for instance, an open normal bisimulation ( ). By Lemma 1, we may reason up to structural congruence to show that processes are open normal bisimilar. This allows us to consider bisimulation candidates that are small, and use structural congruence after a transition to get back in the candidate. Alternatively, one could close the candidate relation under structural congruence, but this would yield much larger candidates, for which related processes would have completely different structures and formal proofs would become much more complicated.

Existential vs. Universal Quantification of freshness. The definitions of input, input normal, and output normal relations all feature universal quantification on some channel and/or variable that have to be fresh. The formalization in Coq has revealed that this condition leads to major problems in a number of proofs, including the one for transitivity of open normal bisimilarity, the proof of Lemma 1, and the proof of Lemma 7; these problems were not explicitly addressed in [13].

Transitivity provides the simplest example: assume $P \sim Q$ and $Q \sim R$, where $\sim$ is a bisimulation relying on a property on fresh variables. To show $P \sim R$, one has to show the property holds for

every variable fresh in P and R. However, we do not know whether every such variable is fresh in Q, so we cannot infer the property from $P \sim Q$ nor $Q \sim R$. Using an existential approach would not work either: the variable witness for $P \sim Q$ may differ from the one for $Q \sim R$. The solution is to show the universal and existential versions of the bisimulations coincide, use the existential version for $P \sim R$ (we need to find a variable), and rely on the universal version for the hypotheses (allowing to freely choose a variable to be fresh for P, Q, and R). Intuitively, the identity of fresh variables does not matter: if one works, any will. We have shown this is the case for the IO, normal, and open normal bisimilarity (🦆) (🦆) (🦆). To achieve this, we have shown that related processes have the same multiset of free variables and the same multiset of channels used for input or output.

Again, as for bisimulations "up-to", we could consider closing the candidate relation under variable freshness, but this would again complicate the formal proof substantially. However, and much more importantly, existentially-quantified IO-bisimulation cannot be avoided in the formalization of the decidability procedure.

4. Formalizing Decidability & Coincidence of Bisimilarities

We now present the formalization of the bisimilarity decidability and bisimilarity coincidence theorems of HOCore. We emphasize that, in order to arrive at those results, it was necessary to prove a number of auxiliary lemmas about structural congruence, substitution, multiple substitution, and transitions that were implicitly considered true in the original paper. Some of these lemmas were purely mechanical, while others required a substantial effort. One of the latter is Lemma 2, presented below and used later in the proof of Lemma 8. Although its correctness is intuitively clear—when substituting several variables with empty messages, the substitution order does not matter—it required a relatively complex proof by complete induction on the length of $\widetilde{X}$.

Lemma 2. Let P be a wf-process, $\widetilde{X}$ and $\widetilde{X'}$ lists of variables, and $\widetilde{M}$ and $\widetilde{M'}$ lists of messages with arguments $\mathbf{0}$. Further, let $\widetilde{X}$ and $\widetilde{X'}$ have no duplicate elements, and let the lengths of $\widetilde{X}$ and $\widetilde{M}$, as well as the lengths of $\widetilde{X'}$ and $\widetilde{M'}$ be equal. Let $(\widetilde{X}, \widetilde{M})$ and $(\widetilde{X'}, \widetilde{M'})$, respectively, denote the list of ordered pairs obtained by joining $\widetilde{X}$ and $\widetilde{M}$, and $\widetilde{X'}$ and $\widetilde{M'}$. Finally, let us assume that for all $X \in \mathsf{fv}(P)$ and arbitrary m, it holds that $(X, \overline{m}\langle\mathbf{0}\rangle) \in (\widetilde{X}, \widetilde{M})$ *if and only if* $(X, \overline{m}\langle\mathbf{0}\rangle) \in (\widetilde{X'}, \widetilde{M'})$. Then, it also holds that $[\widetilde{M}/\widetilde{X}]P = [\widetilde{M'}/\widetilde{X'}]P$. (🦆)

IO-bisimilarity. The first bisimilarity we studied is IO-bisimilarity. Its simplicity lets us show directly that not only it is a congruence, but that it is also decidable.

Lemma 3. The following properties concerning $\sim^{\circ}_{\mathtt{IO}}$ hold:

1. $\sim^{\circ}_{\mathtt{IO}}$ is a congruence: if $P \sim^{\circ}_{\mathtt{IO}} P'$, then: (1) $a[X]_f.P \sim^{\circ}_{\mathtt{IO}} a[X]_f.P'$; (🦆) (2) for all Q, $P \parallel Q \sim^{\circ}_{\mathtt{IO}} P' \parallel Q$; (🦆) and (3) $\overline{a}\langle P\rangle \sim^{\circ}_{\mathtt{IO}} \overline{a}\langle P'\rangle$. (🦆)
2. IO-bisimilarity and existential IO-bisimilarity coincide. (🦆)
3. IO-bisimilarity and IO-bisimilarity up to $\equiv^*$ coincide. (🦆)
4. $\sim^{\circ}_{\mathtt{IO}}$ is a τ-bisimulation. (🦆)
5. $\sim^{\circ}_{\mathtt{IO}}$ is decidable. (🦆) (🦆)

To show decidability, we specify a brute force algorithm that tests every possible transition for the two processes, up to a given depth. (🦆) As IO-bisimilarity testing always results in smaller processes (there is no τ clause in this bisimilarity), we can show that this algorithm decides IO-bisimilarity, if the depth considered is large enough (🦆) (🦆). This allows us to conclude:

```
Theorem IObis_constructive_decidable : forall p q, {p ≈ q} + {~(p ≈ q)}.
```

We should note here that the brute force algorithm would not be applicable to the definition of IO-bisimilarity as stated in Definition 2, because we would need to consider universal quantification on the fresh variable X in the case of an input transition. However, since we have proven the equivalence of universally- and existentially-quantified IO-bisimilarities, it is sufficient to check bisimilarity for only one arbitrary fresh X. More importantly, we strongly emphasize that, although the TLC library is based on classical logic, the decidability procedure that we have formalized is fully constructive.

General Properties of Bisimilarities. Next, we turn to properties of the other bisimilarities that are required to show they all coincide.

Lemma 4. $P \sim^{\star}_{\mathtt{HO}} Q$ *if and only if* for all closed $\widetilde{R}$, $[\widetilde{R}/\widetilde{X}]P \sim_{\mathtt{HO}} [\widetilde{R}/\widetilde{X}]Q$, where $\widetilde{X} = \mathsf{fv}(P) \cup \mathsf{fv}(Q)$. (Lemma 4.13 of [13]). (🐓)

While the proof of this claim takes only several lines on paper, its Coq version amounts to more than three hundred lines of code, requiring a number of auxiliary lemmas that appear evident in a pen-and-paper context. These lemmas mostly deal with with permutations of elements inside a list and the treatment of channels in the "opening" of $\sim_{\mathtt{HO}}$.

Lemma 5. The following properties concerning $\sim_{\mathtt{NOR}}$ and $\sim^{\star}_{\mathtt{NOR}}$ hold:

1. Normal bisimilarity and existential normal bisimilarity coincide. (🐓)
2. $P \sim^{\star}_{\mathtt{NOR}} Q$ *if and only if* there exists a list $\widetilde{M}$ of messages on distinct, fresh channels, each carrying $\mathbf{0}$ and such that $[\widetilde{M}/\widetilde{X}]P \sim_{\mathtt{NOR}} [\widetilde{M}/\widetilde{X}]Q$, where $\widetilde{X} = \mathsf{fv}(P) \cup \mathsf{fv}(Q)$. [6] (🐓)

Lemma 6. The following properties concerning $\sim^{\circ}_{\mathtt{NOR}}$ hold:

1. Open normal and existential open normal bisimilarity coincide. (🐓)
2. Open normal bisimilarity and open normal bisimilarity up to $\equiv^{*}$ coincide. (🐓)
3. If $P \sim^{\circ}_{\mathtt{NOR}} Q$, then P and Q have the same multiset of variables and channels. (🐓) (🐓)

The last claim is proven using induction on the combined measure $m(P)$ that takes into account both the structural size of P ($s(P)$) and the number of output actions in it ($o(P)$). This is necessary since an additional input process is produced in the output normal bisimulation clause of $\sim^{\circ}_{\mathtt{NOR}}$. Thus, in the induction step, the size of the process cannot be guaranteed to decrease. However, the number of output actions becomes smaller, and the combined measure suffices.

Coincidence of Bisimilarities in HOCore. We finally present the formalization of the coincidence of bisimilarities. First, we address one of the two most technically challenging lemmas of the development (Lemma 4.15 of [13]) (🐓) (🐓).

Lemma 7. If $(m[X]_f.P') \parallel P \sim^{\circ}_{\mathtt{NOR}} (m[X]_f.Q') \parallel Q$ for some fresh m, then $P \sim^{\circ}_{\mathtt{NOR}} Q$ and $P' \sim^{\circ}_{\mathtt{NOR}} Q'$.

Proof. In order to show the first part of the lemma, we show that the relation

$$\mathcal{S} = \bigcup_{j=1}^{\infty} \left\{ (P, Q) \ : \ \left(\prod_{k \in 1..j} m_k[X_k]_f.P_k \right) \parallel P \sim^{\circ}_{\mathtt{NOR}} \left(\prod_{k \in 1..j} m_k[X_k]_f.Q_k \right) \parallel Q \right\}$$

[6] The proof of this claim, unlike its $\sim_{\mathtt{HO}}$ counterpart, requires the use of the equivalence between normal and existential normal bisimilarity.

is an existential $\sim^{\circ}_{\texttt{NOR}}$-bisimulation, where $\{P_i\}_1^{\infty}$, $\{Q_i\}_1^{\infty}$ are arbitrary processes, $\{m_i\}_1^{\infty}$ are channels fresh with respect to P, Q, $\{P_i\}_1^{\infty}$, and $\{Q_i\}_1^{\infty}$, and each variable X_i is fresh with respect to P_i and Q_i. The proof proceeds as in [13], by examining possible transitions of P and showing that these transitions are matched by Q. To this end, we study processes of the form $(\prod_{k\in 1..j} m_k[X_k]_f.P_k) \parallel P$ (🦜), proving several lemmas about their properties in relation to transitions and structural congruence. The original proof silently relies on the use of existentially-quantified open normal bisimilarity and up-to structural congruence proof techniques for this bisimilarity (up-to techniques are proven, but only for IO-bisimilarity). These assumed results were not trivial to formally establish.

For the second part of the lemma, we have determined that it is not necessary to formalize the informal, highly intuitive procedure to consume open-normal-bisimilar processes, as presented in [13]. Instead, it is sufficient to proceed by induction on $m(P)$ and make use of the already proven first part, reducing the formal proof to a technical exercise. Moreover, we believe that the formalization of such a consumption procedure would have been very difficult and would require a substantial amount of time and effort. This illustrates the insights one can gain through formal proving when it comes to proof understanding and simplification. □

We are now ready to state and prove the coincidence of bisimilarities.

Theorem 1. *The five strong bisimilarities in HOCore coincide.*

Proof. We prove the following implications, the direct corollary of which is our goal: (1) $\sim^{\circ}_{\texttt{IO}}$ implies $\sim^{\star}_{\texttt{HO}}$ (🦜); (2) $\sim^{\star}_{\texttt{HO}}$ implies $\sim^{\star}_{\texttt{CON}}$ (🦜); (3) $\sim^{\star}_{\texttt{CON}}$ implies $\sim^{\star}_{\texttt{NOR}}$ (🦜); (4) $\sim^{\star}_{\texttt{NOR}}$ implies $\sim^{\circ}_{\texttt{NOR}}$ (🦜); (5) $\sim^{\circ}_{\texttt{NOR}}$ implies $\sim^{\circ}_{\texttt{IO}}$ (🦜).

The only major auxiliary claim that needs to be proven here is that an open normal bisimulation also satisfies the output relation clause; this is an immediate consequence of Lemma 7. □

In addition, we have proven that IO-bisimilarity is correct with respect to barbed congruence (🦜), which, given Theorem 1, means that the forms of bisimilarity that we are considering are all correct w.r.t. barbed congruence. The other direction (completeness) involves properties of asynchronous bisimulations that remain to be formalized.

Next, we focus on the lemma that corresponds to the right-to-left direction of Lemma 4.14 of [13], stating that higher order bisimilarity implies IO-bisimilarity. Interestingly, this claim was included in [13] although it is not necessary to prove the coincidence of bisimilarities. Its proof was the most challenging to formalize, and this formalization has led us to several important insights.

Lemma 8. $\sim^{\star}_{\texttt{HO}}$ implies $\sim^{\circ}_{\texttt{IO}}$.

Discussion. It is sufficient to prove that the relation $\mathcal{R} = \{(P,Q) \mid [\widetilde{M}/\widetilde{X}]P \sim^{\star}_{\texttt{HO}} [\widetilde{M}/\widetilde{X}]Q\}$ is an IO-bisimulation, where $\widetilde{X}$ and $\widetilde{M}$ are, respectively, lists of variables and messages carrying $\mathbf{0}$ on fresh channels. We show $\mathcal{R}$ is an input (🦜), an output (🦜), and a variable relation (🦜). The proofs of all three cases in [13] share the same structure that relies on the application of both directions of Lemma 4.

First, the unguarded variables[7] are substituted with processes of the form $\overline{m}\langle\mathbf{0}\rangle$. Then, the remaining free variables are substituted with arbitrary closed processes $\widetilde{R}$, using Lemma 4 left-to-right to show that the result is still in $\mathcal{R}$. Next, the processes do a transition, and it is proposed to apply Lemma 4 right-to-left to conclude. This last step is not justified, as we now explain. The main idea is to show that the bisimulation diagrams are closed under multiple substitution, i.e., that for all processes P and Q, all global variables X, and all closed processes $\widetilde{R}$, given $[\widetilde{R}/\widetilde{X}]P \sim_{\texttt{HO}} [\widetilde{R}/\widetilde{X}]Q$ and $[\widetilde{R}/\widetilde{X}]P \xrightarrow{\alpha} [\widetilde{R}/\widetilde{X}]P'$, there exists a Q' such that $[\widetilde{R}/\widetilde{X}]Q \xrightarrow{\alpha} [\widetilde{R}/\widetilde{X}]Q'$ and $[\widetilde{R}/\widetilde{X}]P' \sim_{\texttt{HO}} [\widetilde{R}/\widetilde{X}]Q'$.

[7] Variables that appear in an execution context, i.e., variables that are not "guarded" by an input.

Then, the idea is to apply Lemma 4 right-to-left to obtain $P' \sim^{\star}_{\texttt{HO}} Q'$. This is accomplished by first showing that there exists an S, such that $[\widetilde{R}/\widetilde{X}]Q \xrightarrow{\alpha} S$ and $[\widetilde{R}/\widetilde{X}]P' \sim_{\texttt{HO}} S$, and then showing that there exists a Q', such that $S = [\widetilde{R}/\widetilde{X}]Q'$. The crucial mistake is that Q' here depends on $\widetilde{R}$, unless proven otherwise. Thus, the requirement for a unique Q' with a universal quantification on $\widetilde{R}$ of Lemma 4 right-to-left is not satisfied, and the lemma cannot be applied.

Our initial attempt at this proof followed this approach, which failed when Coq refused to let us generalize $\widetilde{R}$ since Q' depended on it. We thus proceeded differently, by first substituting every free variable with processes of the form $\overline{m}\langle\mathbf{0}\rangle$, using Lemma 4 left-to-right to show that we remain in the relation. We then applied Lemma 2 and another auxiliary lemma ( ) to finish the proof.

To conclude this summary, we note another error that we have discovered in Section 6.1 of [13], that concerns the axiomatization of HOCore. Namely, in the proof of the cancellation lemma:

For all processes P, Q, R, if $P \parallel R \sim^{\circ}_{\texttt{IO}} Q \parallel R$, then also $P \sim^{\circ}_{\texttt{IO}} Q$.

the authors attempt to re-use the proof from [16], that proceeds by induction on the sum of sizes of P, Q, and R. However, this is not possible, as that proof was designed for a calculus with operators structurally different than the ones used in HOCore. Instead, we need to use the candidate relation $\mathcal{R}$, parameterized by a natural number n, such that $P \mathrel{\mathcal{R}} Q$ at level n if and only if there exists a process R such that $s(P) + s(Q) + s(R) \leq n$ and $P \parallel R \sim^{\circ}_{\texttt{IO}} Q \parallel R$, and prove that this candidate relation is an IO-bisimulation.

5. Related Work

We first discuss alternative approaches to the handling of binding and α-conversion, starting from *Nominal Isabelle* [28, 11, 30]. This extension of the Isabelle proof assistant features nominal datatypes that represent α-equivalence classes. It has been used successfully in a number of research efforts concerning formalization of various calculi that feature binders [17, 2, 29], and taking advantage of it would probably reduce the size of our development. However, since one of our main results is the formalization of the decidability procedure for IO-bisimulation, and since stating decidability in Isabelle is not a trivial task, we have ultimately opted to use Coq.

The locally nameless approach [1, 6] is the one most similar to the one we are using. There, variables are also split into two categories—local and global—but local variables are calculated by using de Bruijn indices instead of a height function. We find this approach to be both equally viable and equally demanding—freshness and well-formedness would both have to be defined, albeit differently, the proofs would retain their level of complexity, and the amount of code required would not be reduced substantially.

In the Higher-Order-Abstract-Syntax (HOAS), binders of the meta-language are used to encode the binding structure of the object language. In some settings, HOAS can streamline the development and provide an elegant solution for dealing with α-conversion, but in the case of process calculi it also brings certain drawbacks. As stated in [10], there are difficulties with HOAS in dealing with metatheoretic issues concerning names, especially when it comes to the opening of processes, to the extent that certain meta-theoretic properties that involve substitution and freshness of names inside proofs and processes cannot be proven inside the framework and must be postulated instead.

In light of everything previously stated in this section and since HOCore does not have a restriction operator, we have decided to use the canonical locally named approach [19] for variables bound by the input operator. We have not yet considered more general approaches for name binding, such as [20]. In fact, our development may easily be adapted to other models for binders, as long as binders remain canonical (α-convertibility is equality).

We now turn to works related to the formalization of process calculi and their properties. To the best of the authors' knowledge, there have been no formalizations of the higher-order π-calculus so far. There are, however, many formalizations of the π-calculus in various proof assistants, including Coq, such as [9] (where the author uses de Bruijn indices) and [10] (where the authors use HOAS). The work closest to ours is the recently published formalization in Isabelle of higher-order Psi-calculi [17], an extension to the higher-order setting of [2]. We have developed our own formalization because we wanted to stay as close to the paper proofs as possible, in particular when it came to the handling of higher-order and the definitions of the many bisimulations involved. Although we feel that translating HOCore and its bisimulations into a higher-order psi-calculus constitutes a very interesting problem, it is beyond the scope of this paper.

A preliminary version of this paper has appeared in [3]. It contained the initial results of the effort, which included the locally nameless technique for dealing with binders, the correction of the semantics, and the results on decidability. With respect to that version, the contributions of this paper are the new and more solid treatment of well-formed processes as records, the proof of the coincidence of bisimilarities, and the discovery and correction of several errors and imprecisions made in [13].

Finally, we have not fully used the automation that Coq offers. In particular, we could benefit from integrating recent libraries that manipulate binary relations algebraically [4].

6. Lessons Learned and Conclusions

The formalization of HOCore in Coq has given us a deeper understanding of both the calculus itself and the level of precision required for proving properties of bisimulations. We have identified one major flaw in the proofs, namely the improper generalization of an hypothesis in the proof of Lemma 8. This flaw was particularly insidious, as it was due to a lapse in the tracking of the context inside which a property is derived. Such tracking is precisely where proof assistants excel. We have also identified two missing results required by some lemmas. The first concerns existentially-quantified bisimulations: certain properties of bisimulations cannot be proven if the variables or channels featured in freshness conditions of bisimulation clauses are universally quantified. We solved this by introducing bisimulations with existential quantification and showing they are equivalent to the universally quantified ones. We have also discovered that existentially-quantified bisimulations are necessary for the formulation of the decidability procedure. The second missing result is about the transitivity of bisimulations, required to prove that open normal and existential open normal bisimilarities coincide. Proving transitivity is not a trivial claim that required the use of reasoning on bisimulations up to structural congruence, which was also missing for some bisimilarities.

Nevertheless, we were able to prove the decidability of IO-bisimilarity, the main coincidence theorem, and their supporting lemmas. We found that some of the lemmas of the paper are actually not needed, such as Lemma 8. Additionally, we significantly simplified some proofs. For instance, the proof of Lemma 7 (Lemma 4.15 in [13]) does not require the consumption procedure described in [13]; induction on the combined measure $m(P)$ is sufficient and more elegant.

As a side effect of our precise treatment of bound variables, we have detected and corrected an error in the original semantics of HOCore, making it more streamlined in the process. Labels for input transitions no longer mention the variable to be substituted, but instead evolve to an abstraction. Since we do not need to lift the binders, we have decided to make these abstractions local. We have also introduced the operation that deletes a variable as a transition in the LTS, removing a special case in the definition of the bisimulations and avoiding the use of structural congruence.

In fact, many choices made during the formalization were guided by the intent to avoid structural congruence. Although it is very convenient to be able to freely change the structure of a process, doing so interferes with local reasoning. There were cases, however, where we could not avoid it, in particular when proving properties of normal bisimulation. In those cases, we have used "up-

to structural congruence" techniques to keep the bisimulation candidates small enough. We have discovered that the discord between the rigid syntax of a formal process and the intuition that it models a "soup" where processes can freely move around and interact with each other made the formalizations of some of the proofs more difficult than anticipated.

We also proved that the co-inductive and set-theoretic definitions of IO-bisimilarity are equivalent, and used them depending on the complexity of the candidate relation at hand.

As for further work, our immediate aim is to formalize the results presented in Section 6 of [13], pertaining to the soundness and completeness of the axiomatization of HOCore. At this point, we have the proof of the existence and uniqueness of prime decomposition of processes (), as well as an initial description of processes in normal form () and a proof that normalization terminates (). Afterwards, we plan to extend the formalization of HOCore with name restriction, before tackling more complex features such as passivation.

In conclusion, we hope that the experience described in this paper will serve as a motivational step towards a more systematic use of proof assistants in the domain of process calculi.

Bibliography

[1] B. Aydemir, A. Charguéraud, B. C. Pierce, R. Pollack, and S. Weirich. Engineering formal metatheory. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 3–15. ACM, Jan. 2008.

[2] J. Bengtson and J. Parrow. Psi-calculi in isabelle. In *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics*, TPHOLs '09, pages 99–114, Berlin, Heidelberg, Aug. 2009. Springer-Verlag.

[3] S. Boulier and A. Schmitt. Formalisation de hocore en coq. In *Actes des 23èmes Journées Francophones des Langages Applicatifs*, Jan. 2012.

[4] T. Braibant and D. Pous. Deciding kleene algebras in coq. *Logical Methods in Computer Science*, 8(1):1–42, 2012.

[5] Z. Cao. More on bisimulations for higher order *pi*-calculus. In *Proc. of FoSSaCS'06*, volume 3921 of *LNCS*, pages 63–78. Springer, 2006.

[6] A. Charguéraud. The locally nameless representation. *Journal of Automated Reasoning*, pages 1–46, 2011. 10.1007/s10817-011-9225-2.

[7] E. Gimenez. *A Tutorial on Recursive Types in Coq*, 1998. Technical Report No. 0221.

[8] G. Gonthier. A computer-checked proof of the four colour theorem. Available on-line at: `http://research.microsoft.com/en-us/um/people/gonthier/4colproof.pdf`, September 2004.

[9] D. Hirschkoff. A full formalisation of pi-calculus theory in the calculus of constructions. In *Proceedings of the 10th International Conference on Theorem Proving in Higher Order Logics*, volume 1275, pages 153–169. Springer, Aug. 1997.

[10] F. Honsell, M. Miculan, and I. Scagnetto. pi-calculus in (co)inductive-type theory. *Theoretical Computer Science*, 253(2):239–285, Feb. 2000.

[11] B. Huffman and C. Urban. A new foundation for nominal isabelle. In M. Kaufmann and L. C. Paulson, editors, *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, pages 35–50. Springer, 2010.

[12] A. Jeffrey and J. Rathke. Contextual equivalence for higher-order pi-calculus revisited. *Log. Meth. Comput. Sci.*, 1(1):1–22, 2005.

[13] I. Lanese, J. A. Pérez, D. Sangiorgi, and A. Schmitt. On the expressiveness and decidability of higher-order process calculi. *Information and Computation*, 209(2):198–226, Feb. 2011.

[14] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[15] The Coq development team. *The Coq proof assistant reference manual*, 2014. Version 8.4.

[16] R. Milner and F. Moller. Unique decomposition of processes. *Theor. Comput. Sci.*, 107(2):357–363, 1993.

[17] J. Parrow, J. Borgström, P. Raabjerg, and J. Åman Pohjola. Higher-order psi-calculi. *Mathematical Structures in Computer Science*, FirstView:1–37, 3 2014.

[18] B. C. Pierce, C. Casinghino, M. Gaboardi, M. Greenberg, C. Hriţcu, V. Sjoberg, and B. Yorgey. *Software Foundations*. Electronic textbook, 2013.

[19] R. Pollack, M. Sato, and W. Ricciotti. A canonical locally named representation of binding. *Journal of Automated Reasoning*, pages 1–23, May 2011. 10.1007/s10817-011-9229-y.

[20] N. Pouillard and F. Pottier. A fresh look at programming with names and binders. In *Proceedings of the fifteenth ACM SIGPLAN International Conference on Functional Programming (ICFP 2010)*, pages 217–228, Sept. 2010.

[21] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, University of Edinburgh, Dept. of Comp. Sci., 1992.

[22] D. Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141–178, dec 1996.

[23] D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theor. Comput. Sci.*, 167(2):235–274, 1996.

[24] B. Thomsen. A calculus of higher order communicating systems. In *Proc. of POPL'89*, pages 143–154. ACM Press, 1989.

[25] B. Thomsen. *Calculi for Higher Order Communicating Systems*. PhD thesis, Imperial College, 1990.

[26] B. Thomsen. Plain CHOCS: A second generation calculus for higher order processes. *Acta Inf.*, 30(1):1–59, 1993.

[27] A. Tiu and D. Miller. Proof search specifications of bisimulation and modal logics for the pi-calculus. *ACM Transactions on Computational Logic (TOCL)*, 11:13:1–13:35, January 2010.

[28] C. Urban. Nominal techniques in isabelle/hol. *J. Autom. Reasoning*, 40(4):327–356, 2008.

[29] C. Urban, J. Cheney, and S. Berghofer. Mechanizing the metatheory of LF. *ACM Trans. Comput. Log.*, 12(2):15, 2011.

[30] C. Urban and C. Kaliszyk. General bindings and alpha-equivalence in nominal isabelle. In G. Barthe, editor, *Programming Languages and Systems - 20th European Symposium on Programming, ESOP 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, pages 480–500. Springer, 2011.