A Lightweight Formalization of the Metatheory of Bisimulation-Up-To

Kaustuv Chaudhuri, Matteo Cimini, Dale Miller

A Lightweight Formalization of the Metatheory of Bisimulation-Up-To

Kaustuv Chaudhuri
Inria & LIX/École polytechnique
kaustuv.chaudhuri@inria.fr

Matteo Cimini
Indiana University
mcimini@indiana.edu

Dale Miller
Inria & LIX/École polytechnique
dale.miller@inria.fr

December 5, 2014

Abstract

Bisimilarity of two processes is formally established by producing a bisimulation relation that contains those two processes and obeys certain closure properties. In many situations, particularly when the underlying labeled transition system is unbounded, these bisimulation relations can be large and even infinite. The *bisimulation-up-to* technique has been developed to reduce the size of the relations being computed while retaining *soundness*, that is, the guarantee of the existence of a bisimulation. Such techniques are increasingly becoming a critical ingredient in the automated checking of bisimilarity. This paper is devoted to the formalization of the meta theory of several major bisimulation-up-to techniques for the process calculi CCS and the π-calculus (with replication). Our formalization is based on recent work on the proof theory of least and greatest fixpoints, particularly the use of relations defined (co-)inductively, and of co-inductive proofs about such relations, as implemented in the Abella theorem prover. An important feature of our formalization is that our definitions of the bisimulation-up-to relations are, in most cases, straightforward translations of published informal definitions, and our proofs clarify several technical details of the informal descriptions. Since the logic behind Abella also supports λ-tree syntax and generic reasoning using the ∇-quantifier, our treatment of the π-calculus is both direct and natural.

1 Introduction

To formally show that two processes are *bisimilar*, we must exhibit a *bisimulation* relating them. A bisimulation is a relation between processes—*i.e.*, a set of pairs of process *expressions*—that progresses to itself (formally defined in Section 2.1). The cost of naively computing a bisimulation can be proportional to its size, so for an effective use of the bisimulation proof method it is important to control the size of the computed relations. The size of a bisimulation depends on the complexity of the underlying transition system; if the transition system is unbounded, bisimulations are typically infinite sets.

One well known and general approach, originally due to Milner [19], of reducing the size of a computed relation is to represent them *up to* a different relation that identifies redundant pairs of process expressions. Only one representative of each equivalence class of redundant pairs needs to be used in the bisimulation relation. Depending on the identification of redundant pairs, we obtain different *bisimulation-up-to techniques*. For instance, if we identify process expressions up to bisimilarity, then the bisimulation up to bisimilarity relation allows process expressions to be freely rewritten to bisimilar expressions in the bisimulation-up-to relation. Such a technique can have a large impact: for instance, every bisimulation between the CCS processes $!!a$ and $!a$ is infinite, whereas the singleton relation between them is a bisimulation up to bisimilarity. In some process calculi, particularly for higher-order π-calculi, even *defining* bisimulation relations can be difficult or impossible without recourse to up-to-techniques. The size reduction also directly improves automating bisimilarity checks, and, can help to produce more compact proof certificates for bisimilarity.

This paper is concerned with providing formal proofs of metatheoretic properties of both CCS and the π-calculus. Of particular importance are proving theorems that guarantee *soundness* for a number of bisimulation-up-to techniques. A bisimulation-up-to relation is sound if it relates only bisimilar processes, *i.e.*, when it is included in the bisimilarity relation. A bisimulation-up-to relation is generally not itself a bisimulation; indeed, the redundant pairs identified by the up-to-techniques need not be coherent with the closure properties of bisimulation. Soundness is not merely of academic interest; without a formal proof of the soundness of a

bisimulation-up-to technique, one cannot build a complete formal proof of the bisimilarity of two processes using the up-to-technique.

A rather interesting aspect of formalizing bisimulation-up-to—and indeed of bisimulation itself—is the nature of formal co-inductive proofs. Many formal proof systems today provide some kind of co-inductive reasoning capabilities, and each system carries its own restrictions and limitations. There is also the mathematical sense of co-induction, usually based on some standard theory such as Tarski's theory of fixed points, that uses mathematical concepts: sets, relations, lattices, and so on. Generally speaking, the informal ("pen-and-paper") notions of bisimulation, such as Milner's original definition, are formulated in the latter mathematical style, so to formalize the notions we are faced with a choice: do we use the informal definitions directly in terms of a formalization of the mathematical concepts, or do we use the built-in notions of relations and fixed points provided by the framework? By way of an analogy, consider two approaches to capturing Peano arithmetic. The first approach starts with ZF set theory, defines the notion of injective and bijective functions, equivalence classes (cardinals) of sets under bijections, finite sets, and finally, natural numbers as finite cardinals. On top of such a theory, one can start to build up a theory of arithmetic. The alternative approach, used by, *e.g.*, Gentzen [11], starts with a stronger proof system, such as the sequent calculus for first-order logic extended with a few simple rules used to account for induction and term equality. Arithmetic can then be attempted directly in such a sequent calculus.

Both formalization choices can be justifiable depending on the intended use of the formalization. In this work, we propose to use, as much as possible, the built-in notions of relations and fixed points provided by the framework, for the following reasons: (1) users of these systems are much more likely to be familiar with the built-in notions than with a particular mathematical library; (2) tools and techniques for automation in these systems are more likely to use the built-in representations; and (3) formalizing bisimulation-up-to, in particular, is an interesting challenge problem for co-inductive reasoning in formal proof systems. We refer to our approach here as *lightweight* in just the same way that the sequent calculus approach to arithmetic is lightweight when compared to the approach based on set theoretic constructions.

Concretely, we use the intuitionistic logic $\mathcal{G}$ [10] which has a proof-theoretic treatment of least and greatest fixpoints among other features. Bisimilarity is represented in $\mathcal{G}$ as the greatest fixpoint of the progress condition for a given transition system. This representation has already been used to formalize the metatheory of bisimilarity, such as the proof that it is a congruence [28]. We build on this approach by defining the *bisimilarity-up-to* relation as a greatest fixpoint of the progress condition augmented by a given up-to-technique. Because the bisimilarity-up-to relation contains every bisimulation-up-to relation, we can formally establish the soundness of the bisimulation-up-to technique by showing that the bisimilarity-up-to relation is included in the bisimilarity relation. This theorem is proved by direct co-induction on the greatest fixpoint definitions in the Abella implementation [28] of $\mathcal{G}$. It is worth remarking that such lightweight approaches to formalization have surprising consequences at times: for example, the bisimulation we define below for the π-calculus is *open* bisimulation. If, however, we change our formalized logic from intuitionistic to classical logic, the *same* specification of bisimulation becomes late bisimulation [27].

One of the main benefits of this syntactic approach to metatheory is that we can exploit the features of the $\mathcal{G}$ logical framework to represent, declaratively and succinctly, the rules of the transition system specified in the relational SOS format. The most striking of these features is the use of *λ-tree syntax* [15] (a version of *higher-order abstract syntax*) to represent process expressions with binding constructs, such as the name restriction and bound actions in the π-calculus. The logic $\mathcal{G}$ has a sophisticated treatment of the equational theory of λ-terms, built from the notions of nominal abstraction and the ∇-quantifier [10].

The main contributions of this paper are as follows.

- We show that it is indeed possible to formally establish the soundness of significant bisimulation-up-to techniques for standard process calculi even when the transition systems for these calculi is specified directly using an inductive definition and the bisimilarity relations are given transparent co-inductive definitions. (Section 3.)
- For the specific case of CCS, we give a novel formalization of the *faithful contexts* technique that is used to show the substitutivity of single hole contexts, and thereby the soundness of the up-to-context and up-to-bisimilarity-and-context techniques. (Section 4.)
- We show that the definitions can be extended, rather uniformly, to the case of process calculi with mobility and restriction, such as the π-calculus; in particular, our formalization of soundness for bisimulation up to bisimilarity for the π-calculus is completely new. (Section 5.)

Following a standard recipe, the proof of soundness for most techniques is built as a composition of two lemmas: for any bisimilarity-up-to relation $\mathcal{R}$, we first show that $\sim \mathcal{R} \sim \subseteq \sim$ (where $\sim$ is the bisimilarity relation), and then show that $\mathcal{R} \subseteq \sim \mathcal{R} \sim$. The first of these lemmas is proved by a co-induction and its proof is modular: the changes required for particular up-to-techniques are confined to easily identifiable subproofs.

The second of these lemmas has a trivial proof. An indirect benefit of our formalization is that we clarify and make precise a number of points in the published informal descriptions of the bisimulation-up-to technique, while remaining faithful to the spirit of these informal proofs. Indeed, even the structures of the formal and the informal proofs are similar.

The complete Abella development accompanying this paper is available at the following URL [5]:

`http://abella-prover.org/upto/`

An extensive tutorial on Abella is also available [1].

2 Background

2.1 Bisimilarity and Up-To-Techniques

Bisimilarity is generally defined for *labelled transition systems* (LTS). In this section and the next two, we consider in detail the standard LTS for a restriction-free variant of the *calculus of communicating systems* (CCS) [19], where processes ($P, Q, \dots$) and actions (μ) are given by the following grammar (here a ranges over some fixed set of names).

$$\mathsf{Act} \ni \mu ::= a \mid \overline{a} \mid \tau$$
$$\mathsf{Proc} \ni P, Q, \dots ::= \mathbf{0} \mid P \mid Q \mid P + Q \mid \mu.\, P \mid !\, P$$

By standard convention, we shall write simply μ in lieu of $\mu.\,\mathbf{0}$. We use a standard LTS for CCS processes with transitions of the form $P \xrightarrow{\mu} Q$, which is as in [22, Figure 6.1] except using the rules for the replication operator as presented in [25, Table 1.5]. The following are the standard definitions of *bisimulation* and *bisimilarity* for this (and any other) LTS.

Definition 1 (Progression) *Given relations $\mathcal{R}, \mathcal{S} \subseteq \mathsf{Proc} \times \mathsf{Proc}$, we say that $\mathcal{R}$* progresses to *$\mathcal{S}$, written $\mathcal{R} \rightarrowtail \mathcal{S}$, if and only if for every pair of processes P, Q with $P \,\mathcal{R}\, Q$,*

- *whenever $P \xrightarrow{\mu} P'$, there must be a Q' with $Q \xrightarrow{\mu} Q'$ such that $P' \,\mathcal{S}\, Q'$; and*
- *whenever $Q \xrightarrow{\mu} Q'$, there must be a P' with $P \xrightarrow{\mu} P'$ such that $P' \,\mathcal{S}\, Q'$.*

Definition 2 (Bisimulation) *A relation $\mathcal{R} \subseteq \mathsf{Proc} \times \mathsf{Proc}$ is said to be a* bisimulation *if and only if it is a fixpoint of progression (i.e., $\mathcal{R} \rightarrowtail \mathcal{R}$).*

Definition 3 (Bisimilarity) *The union of all bisimulation relations for a given LTS is called* bisimilarity *and is denoted with $\sim$. Two processes P and Q are said to* be bisimilar *if and only if there exists a bisimulation relating them.*

Fact 4 *[25] Every bisimulation is reflexive and symmetric. Bisimilarity is (i) the greatest fixpoint of progression, (ii) the largest bisimulation, (iii) an equivalence and a congruence (i.e., if $P \sim Q$ then, for every process context $\mathcal{C}$, $\mathcal{C}[P] \sim \mathcal{C}[Q]$).*

In order to show that two processes are bisimilar, it suffices to exhibit a set of pairs of process expressions that is a bisimulation. Because bisimulation is a fixpoint of progression, any derivatives of a process pair in a bisimulation must also be in the bisimulation. In order to automatically compute a bisimulation between a pair of processes, we have to exhaustively check all the iterated derivatives of the processes for bisimulation. Worse, since we include the replication operator, the underlying LTS is unbounded and hence the space of derivatives of a process is generally infinite. To make the exhaustive exploration of such spaces tractable, we would have to reason about processes algebraically; however, we are prevented from using any of the algebraic properties of processes—not even structural congruences—to simplify the derivatives being enumerated. Thus, even in simple cases such as $!\,!\,a \sim !\,a$ or $!(a+b) \sim !\,a \mid !\,b$, exhibiting an explicit bisimulation relation is extremely tedious and not mechanizable.

It is therefore natural to consider a refinement of the definition of bisimulation that allows the derivatives to be suitably rewritten. To this extent, Milner introduced the notion of *bisimulation-up-to* relations [19], which are not necessarily themselves bisimulations.

Definition 5 (Up-To-Technique) *A relation $\mathcal{U} \subseteq \mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc}$ is called an* up-to-technique. *Given a relation $\mathcal{R} \subseteq \mathsf{Proc} \times \mathsf{Proc}$, the relation $\mathcal{R}$* up-to *$\mathcal{U}$, written $\mathcal{R}_{\mathcal{U}}$, is such that $P \,\mathcal{R}_{\mathcal{U}}\, Q$ iff there exist $P', Q' \in \mathsf{Proc}$ such that $\mathcal{U}(P, P', Q, Q')$ and $P' \,\mathcal{R}\, Q'$.*

Definition 6 (Up-To) *Given* $\mathcal{R} \subseteq \mathsf{Proc} \times \mathsf{Proc}$ *and* $\mathcal{U} \subseteq \mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc}$*, we say that* $\mathcal{R}$ *is a* bisimulation up to $\mathcal{U}$ *if and only if* $\mathcal{R} \rightarrowtail \mathcal{R}_{\mathcal{U}}$*. The union of all relations that are bisimulations up to* $\mathcal{U}$ *is called the* bisimilarity up to $\mathcal{U}$ *and is depicted* $\sim_{\mathcal{U}}$*.*

Note that the up-to-techniques are suitably general and they can relate their four arguments at will. For many situations, however, it is sufficient to consider only a binary up-to-technique that rewrites each derivative of a bisimilar pair independently, which gives us a simpler variant of the above definitions.

Definition 7 (Binary Up-To) *Given relations* $\mathcal{R}, \mathcal{U} \subseteq \mathsf{Proc} \times \mathsf{Proc}$*, we say that* $\mathcal{R}$ *is a* bisimulation up to $\mathcal{U}$ *iff, for the relation* $\mathcal{V}$ *defined by* $\mathcal{V}(P, P', Q, Q') \triangleq \mathcal{U}(P, P') \wedge \mathcal{U}(Q, Q')$*, it is the case that* $\mathcal{R} \rightarrowtail \mathcal{R}_{\mathcal{V}}$*. Bisimilarity up to* $\mathcal{U}$ *is similarly defined.*

Since the up-to-techniques are not constrained in any way in these definitions, a bisimulation-up-to needs not itself be a bisimulation. However, if it were to be *contained* in a bisimulation, then it would suffice as evidence for bisimilarity.

Definition 8 (Soundness) *We say that an up-to-technique* $\mathcal{U}$ *is* sound *when a bisimulation up to* $\mathcal{U}$ *is contained in a bisimulation; equivalently* $\sim_{\mathcal{U}} \subseteq \sim$*.*

A sound up-to-technique should be understood as identifying redundancies in a bisimulation and allowing the whole relation to be quotiented by that redundancy. Such techniques allow the removal of those pairs of process expressions that are only up-to-variants of the derivatives of another pair in the relation. In other words, any bisimulation-up-to relation for a sound up-to-technique can be suitably *enlarged* by adding all the up-to-variants of the pairs in the relation to get a bisimulation. Conversely, any bisimulation can be seen as just a bisimulation up to syntactic equality of process expressions.

Fact 9 *Let* $= \subseteq \mathsf{Proc} \times \mathsf{Proc}$ *stand for the syntactic equality of process expressions (which we will call the* reflexivity technique*). Then,* $\sim$ *and* $\sim_{=}$ *coincide.*

2.2 Formalization in $\mathcal{G}$ (Abella)

The informal mathematical definitions of Section 2.1 can be directly formalized in the logic $\mathcal{G}$ that has built-in support for reasoning about least and greatest fixpoints [10]. The proof theory of $\mathcal{G}$ has been well developed over a sequence of papers [10, 18] and will not be revisited here. Instead, we will directly use the implementation of $\mathcal{G}$ in the Abella theorem prover [28]. Since the SOS rules of CCS and the π-calculus are representable as Horn clauses, they can be accommodated as ordinary inductive definitions in Abella.[1] As a notational convention, all Abella concrete syntax is depicted using a monospaced font.

The following signature represents processes and actions with the concrete types `proc` and `action` respectively.

```
Kind name,action,proc type.

% Action constructors
Type tau          action.
Type up,dn        name -> action.

% Process constructors
Type null         proc.
Type plus,par     proc -> proc -> proc.
Type act          action -> proc -> proc.
Type repl         proc -> proc.
```

Actions consist of τ actions, represented by `tau`, and dual pairs of output and input actions, represented by the `up` and `dn` constructors, respectively. The abstract type `name` is used to draw the names for actions. Processes are constructed in the obvious way; as an example, the process $!(a.\overline{b})|b$ is encoded by the term `par (repl (act (up a) (act (dn b) null))) (act (up b) null)`, if we add constants `a,b` of type `name` to the signature to represent the free names of the process $!(a.\overline{b}) \mid b$.

The labelled transition system (LTS) for CCS is encoded as an inductive definition `one` for single steps of the transition, shown in Figure 1. The definition is syntax-directed on the input processes and the actions. Note that the cases of the definition are overlapping, so the `one` relation is non-deterministic. For replication, in particular, there are two possible transitions, one where the replicated process makes a visible action (the sixth clause) and the other where two copies of the same process interact internally to expose only a `tau` action (the final clause). This presentation of replication, which is taken from [25, Table 1.5], differs somewhat from Milner's original formulation, which would have replaced these two clauses with the following single clause:

[1] We will *not* need to use Abella's *two-level logic* support in this paper.

```
Define one : proc -> action -> proc -> prop by
  one (act A P) A P
; one (plus P1 P2) A Q                      := one P1 A Q
; one (plus P1 P2) A Q                      := one P2 A Q
; one (par P Q) A (par P1 Q)                := one P A P1
; one (par P Q) A (par P Q1)                := one Q A Q1
; one (repl P) A (par (repl P) Q)           := one P A Q
; one (par P Q) tau (par P1 Q1)             := exists X, one P (up X) P1 /\ one Q (dn X) Q1
; one (repl P) tau (par (repl P) (par Q R)) := exists X, one P (up X) Q /\ one P (dn X) R.
```

Figure 1: The LTS for CCS as a definition

```
one (repl P) A Q := one (par P (repl P)) A Q.
```

We find it easier to work with the presentation in Figure 1. The two styles of definition agree up to structural congruence.

We can then directly define the full bisimilarity-up-to relation by co-induction.

```
CoDefine bisim_up_to :
  (proc -> proc -> proc -> proc -> prop) ->
  (proc -> proc -> prop) by
bisim_up_to Tech P Q :=
  (forall A P1, one P A P1 ->
   exists Q1, one Q A Q1 /\
   exists P2 Q2, Tech P1 P2 Q1 Q2 /\ bisim_up_to Tech P2 Q2) /\
  (forall A Q1, one Q A Q1 ->
   exists P1, one P A P1 /\
   exists P2 Q2, Tech P1 P2 Q1 Q2 /\ bisim_up_to Tech P2 Q2).
```

The definition of `bisim_up_to` is higher-order: its first argument is the up-to-technique, which is used to rewrite the derivatives of the two steps. The reasoning logic $\mathcal{G}$ of Abella is a first-order logic, so allowing this kind of higher-order definition requires some explanation. In Abella, such higher-order definitions are allowed only if *both* of the following conditions are met:

- The higher-order arguments, *i.e.*, the arguments with type containing `prop`, must always be the same universally quantified variable in both head and every recursive occurrence in the body of every clause.
- Such arguments can, moreover, not be used in any subformula to the left of the implication in the body.

These restrictions amount to imposing a *module-like protocol*, wherein every instance of the defined atom might be seen as being produced, uniformly, by instancing a common schematic form. In the rest of this paper, we will use the following notations for the types to avoid spacing problems:

`proc2` for `proc -> proc -> prop`
`proc4` for `proc -> proc -> proc -> proc -> prop`

As an example of an up-to-technique, we can consider reflexivity, which when used as an argument to `bisim_up_to` gives us the usual definition of bisimilarity (*cf.* Fact 9).

```
Define refl_t : proc4 by
  refl_t P P Q Q.
CoDefine bisim : proc2 by
  bisim P Q := bisim_up_to refl_t P Q.
```

We can then define the soundness of a given up-to-technique as follows.

```
Define is_sound : proc4 -> prop by
  is_sound Tech := forall P Q,
    bisim_up_to Tech P Q -> bisim P Q.
```

Abella will emit a warning about this definition of `is_sound` that the higher-order argument variable `Tech` is being used to the left of an implication. While the definition itself does not affect the consistency, it is easy to use such definitions to defeat the syntactic *stratification* checker of Abella that guarantees that all inductive or co-inductive definitions indeed denote a corresponding least or greatest fixed point. Therefore, Abella will prevent the user from using `is_sound` to construct any other definitions; it can only be used in the sense of a syntactic macro, with no associated induction or co-induction principles. Practically, these restrictions on higher-order arguments amount to preventing the definitions of up-to-techniques that themselves make use of particular properties of the up-to-technique (such as soundness).

3 Up-To-Bisimilarity

The bisimilarity relation $\sim$ is itself an excellent example of a binary up-to-technique. In this section we discuss in detail our formulation of bisimilarity up to bisimilarity and its soundness proof in Abella.

Intuitively, a bisimulation up to bisimilarity allows us to perform the co-inductive step in reasoning about bisimulation not just with the derivatives of the starting pair of processes, but also with any pair that is pointwise bisimilar to them. As we have seen in the previous section, in order to formulate this technique all we need to do is to define the appropriate predicate that encodes the bisimilarity-up-to technique, which is straightforward: For the sake of clarity, let us be explicit about the associated up-to-technique.

Definition 10 (Up-To-Bisimilarity) *The relation* $\mathcal{B} \subseteq \mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc}$ *is determined by*

$$\mathcal{B}(P_1, P_2, Q_1, Q_2) \triangleq P_1 \sim P_2 \wedge Q_1 \sim Q_2.$$

This is directly represented in Abella:

```
Define bisim_t : proc4 by
  bisim_t P1 P2 Q1 Q2 :=
    bisim P1 P2 /\ bisim Q1 Q2.
```

To prove the soundness of $\mathcal{B}$, *i.e.*, that $\sim_{\mathcal{B}} \subseteq \sim$, one typically uses two lemmas, one establishing that $\sim \sim_{\mathcal{B}} \sim \subseteq \sim$, and the other that $\sim_{\mathcal{B}} \subseteq \sim \sim_{\mathcal{B}} \sim$. In fact, this kind of factoring of a soundness proof for an arbitrary up-to-technique is common enough that we first define the two lemmas generically.

```
Define is_sound_fst : proc4 -> prop by
  is_sound_fst Tech := forall P Q,
    (exists R S, bisim P R /\
       bisim_up_to Tech R S /\
       bisim S Q) -> bisim_up_to Tech P Q.

Define is_sound_snd : proc4 -> prop by
  is_sound_snd Tech := forall P Q,
    bisim_up_to Tech P Q ->
      (exists R S, bisim P R /\
         bisim_up_to Tech R S /\
         bisim S Q).
```

For the specific up-to-technique $\mathcal{B}$, we can then show soundness directly by instantiating `Tech` with `bisim_t`.

```
Theorem bisim_sound_fst :
  is_sound_fst bisim_t.
Theorem bisim_sound_snd :
  is_sound_snd bisim_t.
Theorem bisim_sound : is_sound bisim_t.
```

We omit full Abella proofs in this paper because they can be found in [5]. Instead, in the rest of this section we will discuss some details of these proofs. We begin with the main soundness theorem `bisim_sound`.

`bisim_sound`: To illustrate how proofs are written in Abella, let us begin with the last of these theorems, a straightforward consequence of `bisim_sound_fst` and `bisim_sound_snd`. In order to express this composition in Abella, we first have to *unfold* the definitions `is_sound_fst`, `is_sound_snd`, and `is_sound` to expose the implications. Abella does not unfold definitions automatically because unfolding is a core component of its treatment of induction and co-induction measures.

Since Abella's reasoning logic is first-order, one cannot prove this composition generically in a manner such as:

```
forall (Tech : proc4),
  is_sound_fst Tech -> is_sound_snd Tech ->
    is_sound Tech.
```

The type of `Tech` contains `prop`, but `forall` can only quantify over terms whose types do not contain `prop`. As a consequence, every instance of this theorem has to be stated and (re-)proved separately. For such usage patterns a real module system in Abella would have avoided the repetition. Fortunately, the repeated proof is tiny and identical for every case, even if the associated definition of the step or the bisimulation-up-to relations were to change.

`bisim_sound_snd`: The proof of this lemma only relies on the reflexivity of bisimilarity, which is the lemma `bisim_refl` in our formalization [5] (*cf.* Fact 4). After expanding the definition of `bisim_sound_snd` and assuming the hypothesis `bisim_up_to bisim_t P Q`, we are left with the obligation

```
exists R S, bisim P R
  /\ bisim_up_to bisim_t R S
  /\ bisim S Q
```

At this point, we instantiate `R` and `S` with `P` and `Q` respectively, and appeal to reflexivity of bisimilarity in order to discharge the two conjuncts `bisim P P` and `bisim Q Q`. The remaining conjunct is exactly the assumption of the lemma. It hardly merits remarking that this proof too is a direct representation of its informal version and can be employed for different techniques and different equivalence relations, as long as they are reflexive.

`bisim_sound_fst`: The core component of the soundness theorem is the lemma `bisim_sound_fst`, which we now discuss in some detail. The proof must proceed by co-induction, which is implemented in Abella using a tactic that introduces a *co-inductive hypothesis* (labelled `CH`):

```
forall P Q,
  (exists R S, bisim P R
     /\ bisim_up_to bisim_t R S /\ bisim S Q)
  -> bisim P Q +
```

The `+` at the end is a *co-inductive size annotation* which asserts that the atom `bisim P Q` has resulted from at least one unfolding of the conclusion of the theorem `bisim_sound_fst`. In order to enforce this restriction, the goal of the theorem is rewritten to:

```
forall P Q,
  (exists R S, bisim P R
     /\ bisim_up_to bisim_t R S /\ bisim S Q)
  -> bisim P Q #
```

where the `#` annotation asserts that co-inductive progress has *not* been made.

We can now introduce the variables and antecedents of the goal (using the `intros` tactic), leaving the residual goal `bisim P Q #`. In order to finish the proof, we must take at least one co-inductive step by unfolding the (co-)definition of `bisim`, which will change the size annotation from a `#` to a `+` and thereby enables appeals to the `CH`. (The soundness of this approach to co-induction is addressed in [9].)

Unfolding the definition of `bisim` (and then the resulting `bisim_up_to refl_t`) will result in two subgoals corresponding to steps from `P` and from `Q` respectively. We discuss only one of these here as the other is symmetrical and similar. The goal here is:

```
forall A P1, one P A P1 ->
  (exists Q1, one Q A Q1 +
     /\ (exists P2 Q2, refl_t P1 P2 Q1 Q2 +
           /\ bisim P2 Q2 +)).
```

Observe that the `+` annotation propagates to the three conjuncts. At this point we can introduce the antecedent `one P A P1` as a new hypothesis. The rest of the proof consists of an alternation of two different kinds of reasoning: (1) for each action `A`, we must find a corresponding instantiation for `Q1` that makes `one Q A Q1` true, and (2) for each such `Q1`, we must relate it to the derivative `P1` in order to make the `CH` applicable.

Finding the derivative `Q1`: Let us recall that we already assumed the following hypotheses (with names written with colons to the left) before we initially unfolded the goal `bisim P Q #`:

```
H1 : bisim P R
H2 : bisim_up_to bisim_t R S
H3 : bisim S Q
```

where `R` and `S` are the variables introduced by destructing the `exists` in the antecedent. From these assumptions, we can make the following chain of inferences: for each action taken by `P`, there will be a corresponding action from `R` (by `H1`), which in turn implies the same action from `S` (by `H2`), and then finally from `Q` (by `H3`). For the most part, this chain of inferences merely requires unfolding the definitions of `bisim` and `bisim_up_to` (which is achieved using the `case` tactic), and then plumbing the bodies together. At this point, we will be able to instantiate `Q1` in the goal and discharge the first conjunct that results. Discharging the second conjunct about `refl_t` merely requires instantiating `P2` and `Q2` with `P1` and `Q1` respectively. This leaves just the third conjunct, `bisim P1 Q1 +`.

Relating the derivatives: Note that this goal now matches the head (*i.e.*, the rightmost atom in the chain of implications) of the `CH`. We can therefore *backchain* the `CH` to the goal giving us the residual subgoal:

```
exists R1 S1, bisim P1 R1
   /\ bisim_up_to bisim_t R1 S1
   /\ bisim S1 Q1
```

It is important to note that we used the `CH` to reason about the *derivatives* of the original processes, which is why the `CH` needed to be universally closed.

Each of these conjuncts is nearly found among the assumptions that resulted earlier from unfolding `H1`, `H2`, and `H3`. However, they are not exactly present and we need to appeal to additional properties of bisimilarity that need to be established earlier. In particular, we will need to appeal to both symmetry and transitivity of bisimilarity (*cf.* Fact 4) to complete the chain of inferences. This is not surprising, as bisimilarity up to bisimilarity is the greatest fixpoint of a progression from a relation $\mathcal{R}$ to $\sim \mathcal{R} \sim$. (Only transitivity is explicitly mentioned as a necessary condition in [22, Lemma 6.2.3]; this point is not as relevant to the up-to-bisimilarity technique, since bisimilarity is an equivalence, but it becomes important for other techniques.)

Discussion: An important feature of this proof is that it is *schematic* in two orthogonal senses. First, were we to consider a different up-to-technique, only the reasoning about relating the derivatives needs to be replaced. Second, replacing the LTS for CCS with a different one preserves the general structure of the proof; only the number of cases might differ depending on the definition of bisimilarity and the step relation of the LTS. The essence of the soundness of each up-to-technique is just reasoning about relating the derivatives of two parallel steps, which is the only component of the soundness proof that differs. It is readily apparent in reading the accompanying formal development available at [5] that we are able to maintain this schema when proving the corresponding theorems for open bisimilarity of the π-calculus and even for weak bisimilarity of CCS and the π-calculus.

In [5] we offer some examples of using bisimilarity up to bisimilarity to show that two CCS processes are bisimilar. Specifically, we have the full proof of $!!a \sim !a$ and $!(a + b) \sim !a \mid !b$ using the up-to-bisimilarity technique and its soundness. It is nearly impossible to construct a bisimulation for these cases directly as the relations are infinite and have a rather convoluted internal structure. On the other hand, we can construct the evidence for them being bisimilar up to bisimilarity formally by a direct co-inductive proof.

4 Up-to Context

While bisimulation up to bisimilarity is an excellent tool for reducing the size of the computed relations, and is perhaps the best known example of an up-to-technique [19], it still does not allow compositional reasoning about bisimulation. The *up-to-context* technique [22, Section 6.2.2] is better suited for this kind of reasoning: it uses contextual closure to factor out a common context in the derivatives and hone on the relevant subprocesses.

A process context $\mathcal{C}$ is a process expression with a single hole occurring in the position of a process subexpression, and $\mathcal{C}[P]$ represents that process expression that results from replacing the hole with P. We represent contexts in Abella using the type `cx` with the following signature.

```
Kind cx type.
Type hole          cx.
Type act_d         action -> cx -> cx.
Type plus_l,par_l  cx -> proc -> cx.
Type plus_r,par_r  proc -> cx -> cx.
```

Context substitutions are defined inductively using the ternary relation `at`, with the interpretation `at C P Q` if and only if $\mathcal{C}[P] = Q$ when `C`, `P`, `Q` stand respectively for $\mathcal{C}$, P, Q.

```
Define at : ctx -> proc -> proc -> prop by
  at hole P P
; at (plus_l C R) P (plus Q R) := at C P Q
; at (plus_r R C) P (plus R Q) := at C P Q
; at (par_l C R)  P (par Q R)  := at C P Q
; at (par_r R C)  P (par R Q)  := at C P Q
; at (act_d A C)  P (act A Q)  := at C P Q.
```

Note that we do not allow the context hole to be present under a replication, so $!a.[\,]$ is not an example of a valid context; this restriction is explained below.

We follow closely the definition in [22, Section 6.2.2] in our definition of the up-to-context technique.

Definition 11 (Up-To-Context) *The relation* $\mathcal{K} \subseteq \mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc}$ *is given as follows:*

$$\mathcal{K}(P_1, P_2, Q_1, Q_2) \triangleq \exists \mathcal{C}.\, \mathcal{C}[P_2] = P_1 \wedge \mathcal{C}[Q_2] = Q_1.$$

Note that this is the first place where it is important that both pairs in a bisimulation are rewritten *at once*, since we must factor out a common context; thus, it is not possible to define this relation using binary up-to-techniques. This technique is readily formalized in Abella.

```
Define context_t : proc4 by
  context_t P1 P2 Q1 Q2 :=
    exists C, at C P2 P1 /\ at C Q2 Q1.
```

The proof of soundness of the up-to-context technique is a bit more involved than that of up-to-bisimilarity. Since a common context is factored out of the derivatives of the two candidate processes, it is important to show that this kind of factoring is coherent with the bisimilarity relation itself. In fact, we need a stronger property: we need to know that $\sim_{\mathcal{K}}$ (*i.e.*, `bisim_up_to context_t`) is *substitutive*, that is, whenever $P \sim_{\mathcal{K}} Q$ that also $\mathcal{C}[P] \sim_{\mathcal{K}} \mathcal{C}[Q]$ for any context $\mathcal{C}$. We write this as follows in Abella:

```
Define substitutive : proc2 -> prop by
  substitutive_rel Rel :=
    forall P1 P2 C Q1 Q2,
      at C P1 Q1 -> at C P2 Q2 ->
      Rel P1 P2 -> Rel Q1 Q2.
```

Faithful contexts: We cannot directly show by co-induction that `substitutive (bisim_up_to context_t)` because we need to argue about the actions of a process in an arbitrary context and how it relates to the actions of the process itself. Unless contexts are well-behaved, there is no reason to think that these sets of actions have any relation. Fortunately, in the case of CCS we can show that the contexts are *faithful* [23], which allows us to characterize precisely the actions of a process in a context.

Definition 12 (Faithful Contexts) *A set of contexts $\mathcal{S}$ is said to be* faithful *if for every $\mathcal{C} \in S$, processes P, R, and label μ, if $\mathcal{C}[P] \xrightarrow{\mu} R$, then:*

- *there is a $\mathcal{C}' \in \mathcal{S}$ such that $R = \mathcal{C}'[P]$, and $\mathcal{C}[Q] \xrightarrow{\mu} \mathcal{C}'[Q]$ for every process Q; or*
- *there is a $\mathcal{C}' \in \mathcal{S}$, a process P', and a label μ' such that $P \xrightarrow{\mu'} P'$ and $R = \mathcal{C}'[P']$, and $\mathcal{C}[Q] \xrightarrow{\mu} \mathcal{C}'[Q']$ for all processes Q, Q' for which $Q \xrightarrow{\mu'} Q'$; or*
- *R is independent of P, i.e., for every process Q it is the case that $\mathcal{C}[Q] \xrightarrow{\mu} R$.*

The third option in the definition above is needed to account for the possibility that the hole in a context is in a summand that will be discarded in a step. The definition of faithful contexts in [23] handled this situation in a slightly different form, using contexts with *at most one hole*. It turns out that allowing for no holes in contexts is unwieldy as any theorem involving contexts now has to either account for this possibility or rule it out somehow. We therefore stay with our simpler case of contexts with exactly one hole and just suitably alter the definition of faithful contexts. The two variants of the definition are equivalent, and either definition would have sufficed for our ultimate goal of showing substitutivity of the up-to-context technique.

We prove that all CCS contexts are faithful in Abella directly by induction on the structure of contexts. Fortunately, since `at` is defined in a syntax-directed way on contexts, it is sufficient to induct on derivations of `at` directly. The Abella theorem and the start of its proof is then as follows.

```
Theorem ctx_faithful : forall C P P0 A R,
  at C P P0 -> one P0 A R ->
     (exists CC, at CC P R /\
        (forall Q Q0, at C Q Q0 ->
           exists RR, one Q0 A RR
             /\ at CC Q RR))
  \/ (exists CC PBP B, one P B PBP
        /\ at CC PBP R
        /\ (forall Q QBQ, one Q B QBQ ->
              (forall Q0, at C Q Q0 ->
                 exists RR, one Q0 A RR
                   /\ at CC QBQ RR)))
  \/ (forall Q Q0, at C Q Q0 -> one Q0 A R).
%% induct on: at C P P0
induction on 1. intros. case H1.
```

The proof proceeds in a systematic fashion by induction on the structure of contexts, which is uniquely determined by the cases of the `at` relation. We require one important lemma about context *concatenation*: if $\mathcal{C}_1[P_1] = P_2$ and $\mathcal{C}_2[P_2] = P_3$, then there exists a context $\mathcal{C}_3$ such that $\mathcal{C}_3[P_1] = P_3$.

```
Theorem concat_ctx2 :
  forall C1 C2 P1 P2 P3 Q1 Q2 Q3,
    at C1 P1 P2 -> at C2 P2 P3 ->
    at C1 Q1 Q2 -> at C2 Q2 Q3 ->
    exists C3, at C3 P1 P3 /\ at C3 Q1 Q3.
```

Our proof is greatly simplified by the fact that the `at` relation is deterministic in its second and third arguments, *i.e.*, that $\mathcal{C}[P_1] = \mathcal{C}[P_2]$ if and only if $P_1 = P_2$. If we were to use contexts with at most one hole, then we would need an additional nested induction to rule out the no-hole case.

One important note is that if we were to attempt to extend the definition of contexts to allow holes under replication, then the contexts would not remain faithful. In fact, this remains the case even if replication is guarded by action prefixes. For a simple example,[2] consider the context $\mathcal{C} = !\,a.[\,]$. Then, for any process P, we have $\mathcal{C}[P] \xrightarrow{a} \mathcal{C}[P] \mid P$, but the derivative cannot be represented in the form $\mathcal{C}'[P]$. This limitation may be relaxed somewhat with the use of variadic contexts, but it is unclear how, if at all, the notion of faithful contexts would extend to it.

Proving substitutivity: We can then return to showing that the `context_t` technique is substitutive.

```
forall P1 P2 C Q1 Q2,
   at C P1 Q1 -> at C P2 Q2 ->
     bisim_up_to context_t P1 P2 ->
       bisim_up_to context_t Q1 Q2.
```

We proceed here by co-induction. After unfolding the definitions in the conclusion, we then immediately appeal to `ccs_faithful` to obtain a result about the first assumption, `at P0 C P`. This gives us two cases. In the first case, the process `P` stays unchanged during `one P0 A R`, in which case we directly obtain the required result for `Q0` from the definition of faithful contexts. In the second case, we proceed by case-analysis for the possible transitions of `P`, and in each case, we use the definition of faithful contexts to construct a matching step for `Q`. The proof here critically relies on the fact that `at` and `concatenate` are deterministic to correlate the contexts that are existentially quantified in the definition of faithful contexts to those resulting from unfolding `bisim_up_to context_t`.

Soundness: Once we know that `bisim_up_to context_t` is substitutive, it is then a simple corollary that `context_t` must be sound. From the definition of `context_up_to`, we know that there is a common context in the derivatives that is factored out and the remainder are bisimilar up to context. By substitutivity, the derivatives must therefore themselves be bisimilar up to context. The required result follows trivially from the co-inductive hypothesis.

```
Theorem context_sound : is_sound context_t.
```

The full details of the proof can be found in [5].

Combining up-to-bisimilarity and up-to-context. The nature of co-induction in $\mathcal{G}$ and Abella is general enough to allow for the combination of multiple up-to-techniques. In this paper we do not formalize the general theory of when two up-to-techniques can be combined, as done informally in [22] and largely formalized in Coq in [20, 21]. Instead, we illustrate an important combination of the up-to-bisimilarity and up-to-context techniques (*cf.* [22, Definition 6.2.11]).

Definition 13 (Up-To-Bisimilarity-And-Context) *The relation $\mathcal{BK}$, a subset of* $\mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc}$*, is as follows.*

$$\mathcal{BK}\,(P_1, P_3, Q_1, Q_3) \triangleq \exists P_2, Q_2.\ \mathcal{B}\,(P_1, P_2, Q_1, Q_2) \wedge \mathcal{K}(P_2, P_3, Q_2, Q_3).$$

This technique is readily formalized in Abella:

```
Define bisim_context_t : proc4 by
  bisim_context_t P1 P3 Q1 Q3 :=
    exists P2 Q2, bisim_t P1 P2 Q1 Q2
      /\ context_t P2 P3 Q2 Q3.

Theorem sound_bisim_context_t :
  is_sound bisim_context_t.
```

We can prove this relation sound by a combination of the approaches of Section 3 and this section, by using the factorization to handle reasoning up to bisimilarity and the congruence of bisimilarity to reason up to context. The proof is omitted here for space reasons, but can be found in [5].

Examples: Also in [5] are the following illustrative examples: $!\,!\,a.P \sim !\,a.P$ and $!(a.P + b.P) \sim !\,a.P \mid !\,b.P$, for every process P. This universal quantification over the continuation processes requires both up-to-context and up-to-bisimilarity reasoning, unlike the examples at the end of Section 3 where up-to-bisimilarity sufficed.

[2] Suggested by Damien Pous in private communication

5 From CCS to the π-Calculus

We now summarize how the definitions and theorems above for CCS can be lifted to accommodate the binding structures of the π-calculus. It is striking how little the formalization in Abella needs to change to accommodate such binding structures. Reasoning about names and name restriction in the π-calculus is directly achieved using $\mathcal{G}$'s nominal abstraction [10], ∇-quantifier [18], and λ-tree syntax [15]. While we cannot given a formal presentation of these extensions to logic, their accumulation into one logic yields a system in which names of bound variable are not semantically interesting and are, in fact, impossible to accept. Instead of using the names of binders, Abella and $\mathcal{G}$ have flexible and more declarative ways to treat binders that involve the nesting of term-level λ-abstractions and the $\forall$, $\exists$, and ∇ quantifiers.

Processes in the π-calculus are as follows (where $x, y, \ldots$ range over names):

$$P, Q, \ldots ::= \mathbf{0} \mid P \mid Q \mid P + Q \mid !\,P \mid \tau.\,P \mid x\,(y)\,.\,P \mid \overline{x}\,\langle y\rangle\,.\,P \mid (\nu x)\,P\ .$$

The first four constructs are common to CCS, while the remaining ones—τ actions, input, output, and name restriction—are different for the π-calculus. These new constructs are encoded in $\mathcal{G}$ using λ-tree syntax as follows.

```
Type taup  proc -> proc.
Type out   name -> name -> proc -> proc.
Type in    name -> (name -> proc) -> proc.
Type nu    (name -> proc) -> proc.
```

As an example, the process $(\nu x)\,(!(x\,(y)\,.\,P + Q) \mid \overline{x}\,\langle k\rangle\,.\,R)$ is represented as:

```
nu x\ (par (repl (plus (in x P) Q))
           (out x k R))
```

where `k`, `P`, `Q`, `R` represent k, P, Q, R respectively. The form `x\ T` is concrete syntax in Abella—borrowed from λProlog [15]—for the λ-term $\lambda x.T$, so $(\nu x)\,P$ is written simply as `nu x\ P`, which parses as `nu (x\ P)` by Abella's precedence rules. Note that the type of `P` is `name -> proc`, so the bound name y is not explicitly mentioned in the encoding; it could be made explicit by writing the input-prefixed process as `in x (y\ P y)`, but in this case `P y` indicates an explicit *dependency* of P on the input variable y. If we were to write it instead as `in x (y\ P)`, then the lack of dependency of P on y would be interpreted to mean that y cannot be free in P. This kind of intrinsic encoding of syntactic side-conditions is a feature of λ-tree syntax that we will rely on pervasively.

Indeed, the key feature of the LTS for the π-calculus is the notion of a *bound step*, which is usually treated in informal presentations of the π-calculus (such as in [25]) using a notion of α-variance over the traces and derivatives. In $\mathcal{G}$ and Abella we can avoid this distraction by using λ-tree syntax also for the bound actions. Concretely, the (late) transitions for the π-calculus are formalized in terms of the mutually inductive definitions of bound actions (`oneb`) and other steps (`one`), of which only the interesting cases are shown below.

```
Define one  : proc -> action -> proc -> prop,
       oneb : proc -> (name -> action) ->
                      (name -> proc) -> prop by
   /* closed_actions */
  one (taup P) tau P
; one (out X Y P) (up X Y) P
      ...
   /* communication */
; one (par P Q) tau (par (M Y) T) :=
    exists X, oneb P (dn X) M
           /\ one Q (up X Y) T
; one (par P Q) tau (par R (M Y)) :=
    exists X, oneb Q (dn X) M
            /\ one P (up X Y) R
      ...
   /* bound actions */
; oneb (in X M) (dn X) M
; oneb (plus P Q) A M := oneb P A M
; oneb (plus P Q) A M := oneb Q A M
; oneb (par P Q) A (x\ par (M x) Q) :=
    oneb P A M
; oneb (par P Q) A (x\ par P (N x)) :=
    oneb Q A N
; oneb (nu M) (up X) N :=
    nabla y, one (M y) (up X y) (N y)
; oneb (repl P) A (x\ par (repl P) (M x)) :=
    oneb P A M.
```

The cases for communication highlight how bound input actions match up with (closed) output actions.

The definition of bisimilarity-up-to for the π-calculus extends the related definition for CCS by incorporating additional cases to deal with bound steps.

```
CoDefine bisim_up_to : proc4 -> proc2 by
bisim_up_to Tech P Q :=
   (forall A P1, one P A P1 ->
      exists Q1, one Q A Q1 /\
        exists P2 Q2, Tech P1 P2 Q1 Q2
                    /\ bisim_up_to Tech P2 Q2)
/\ (forall X P1, oneb P (dn X) P1 ->
      exists Q1, oneb Q (dn X) Q1 /\
        exists P2 Q2, forall w,
          Tech (P1 w) (P2 w) (Q1 w) (Q2 w) /\
          bisim_up_to Tech (P2 w) (Q2 w))
/\ (forall X P1, oneb P (up X) P1 ->
      exists Q1, oneb Q (up X) Q1 /\
        exists P2 Q2, nabla w,
           Tech (P1 w) (P2 w) (Q1 w) (Q2 w) /\
           bisim_up_to Tech (P2 w) (Q2 w))
/\ (forall A Q1, one Q A Q1 ->
      exists P1, one P A P1 /\
        exists P2 Q2, Tech P1 P2 Q1 Q2
                    /\ bisim_up_to Tech P2 Q2)
/\ (forall X Q1, oneb Q (dn X) Q1 ->
      exists P1, oneb P (dn X) P1 /\
        exists P2 Q2, forall w,
           Tech (P1 w) (P2 w) (Q1 w) (Q2 w)
        /\ bisim_up_to Tech (P2 w) (Q2 w))
/\ (forall X Q1, oneb Q (up X) Q1 ->
      exists P1, oneb P (up X) P1 /\
        exists P2 Q2, nabla w,
          Tech (P1 w) (P2 w) (Q1 w) (Q2 w)
       /\ bisim_up_to Tech (P2 w) (Q2 w)).
```

The number of cases in the definition of `bisim_up_to` grows from two to six, two each for closed, bound input, and bound output actions.

Two important features of the continuations of bound actions are highlighted using particular colors in this definition. The continuations of bound inputs (`dn X`) are defined using the `forall`-quantifier: to show that two processes, both of which can do a bound input on the same named channel, are in a given bisimulation, one should show that for every instance (*i.e.*, using the $\forall$ quantifier of $\mathcal{G}$) of the bound value, the two instantiated continuations are also in that bisimulation. Similarly, continuations of bound outputs (`up X`) are treated with the `nabla`-quantifier because showing that two processes, both of which can do a bound output on the same named channel, are in a given bisimulation, requires selecting a fresh new name (using the ∇ quantifier of $\mathcal{G}$) and showing that the two continuations, instantiated with that name, are also in that bisimulation. The adequacy of this encoding of open bisimulation [25] for the π-calculus is shown in [27].

The remaining definitions of the preceding sections can be generalized to handle the π-calculus and the theorems and proofs can be adjusted (by hand) in a straightforward and predictable fashion. Indeed, the definitions of the up-to-techniques does not change at all, and only the proofs of the soundness theorems need to be modified to accommodate the additional cases of bisimilarity-up-to in the π-calculus.

Up-To-Bisimilarity: The proof of soundness for the up-to-bisimilarity technique for the π-calculus is structurally identical to that for CCS (in Section 3, except with three times as many cases. For the most part it was straightforward to adapt the CCS proof to the π-calculus proof by hand. However, this is another instance where having some kind of modular proof language would have been an immense benefit. Much of the argumentation is common across the different cases of bisimulation, but the proof text needs to be manually duplicated.

Nevertheless, a major benefit of using the Abella framework is that none of the overhead of moving from a process calculus with binding manifested in the encodings of syntax. In particular, the usual side conditions that appear in informal presentations of the LTS for the π-calculus, particularly in the *open* and *close* rules for name restrictions, is manifested directly by λ-dependencies. This is best illustrated by the penultimate clause of the `oneb` relation, which we write here in a slightly more explicit form.

```
; oneb (nu (y\ M y)) (y\ up X y) (y\ N y) :=
    nabla y, one (M y) (up X y) (N y)
```

This clause, which implements the *close* rule of the LTS, directly links bound name `y` in the label `up X y` and the continuation process `N y` to the restricted name `y` in the input process. Note here that using a `forall` would be invalid, since we need to know that the restricted name `y` is distinct from all the other free names in `M y`.

Up-To-Context: Extending the soundness result to the up-to-context and up-to-bisimilarity-and-context for the π-calculus is a much more involved proposition. In fact, even defining the notion of a process context in the π-calculus is tricky, because contexts are allowed to capture free names. Some of the free names of P in $\mathcal{C}[P]$ may be captured by input actions prefixes (`in`), while others may be captured by restrictions (`nu`).

A more pressing issue is that the proof of substitutivity of a π-calculus context cannot appeal to the notion of faithful contexts, because that is a purely first-order notion. One alternative would be to use *evolutions* and *initial contexts* as developed in [22], but we leave it to future work.

6 Related Work

The π-calculus has been formalized several times before. An early formalization [12] used Coq to specify and prove properties of strong late bisimilarity: that formalization built formal mechanisms to support notions such as "freshness" within a scope, substitution of names, occurrences of names in expressions, *etc.* A similar approach is followed in [7] but instead of Coq, the nominal set theory of [8] was used to develop those formal mechanisms. An Isabelle implementation of the nominal logic approach was used in [3] to formalize and prove some of the metatheory of the π-calculus. This paper follows the papers [16, 17, 26] which used the λ-tree syntax approach to specification in order to capture the nominal and binding structures of the π-calculus: those formal mechanisms are captured differently using the natural interplay of nested quantifiers and λ-bindings in expressions.

This paper is also not the first attempt to formalize the bisimilarity-up-to technique. The oldest, largest, and possibly the most comprehensive of these formalizations has been done in Coq [20, 21, 4]; indeed, this framework has been set up to tackle the much harder problem of weak bisimilarity-up-to, which is well known to have a rather subtle definition [24, 22]. The main difference from our approach in this present work is that this formalization in Coq builds the theory of bisimulation from scratch, starting from a formalization of the mathematics of relations, fixpoints, and so on. While this allows the framework to remain general, it does not use the co-induction of Coq; in particular, the definition of bisimulation is not a co-fixpoint in Coq. Moreover, the current state of the Coq development uses only illustrative examples of simple LTS, instead of standard ones such as that of CCS or the π-calculus.

Arguably, the implementation of co-recursion in Coq cannot be used directly for bisimulation-up-to (or even bisimulation) because of a rather inflexible syntactic productivity check. The approach of [6] is the most recent example of approaches that try to circumvent this check by means of rewriting techniques. Many such approaches can be seen as instances of a general family of *parametrized co-induction* [13]. Such approaches have at least two limitations currently: first, the encoding of bisimulation is not directly related to the SOS specification of an LTS, and so requires an additional informal step of "compiling" the LTS; and second, this approach has no built-in support for names and binding.

A recent paper [14] shows that reasoning about the up-to-techniques that the present paper addresses for the π-calculus can be transported onto the ordinary theory of bisimulation-up-to that targets calculi with no binding. To enable this result, the authors offer an accurate encoding from the richer LTS of the π-calculus into a version of it that avoids binding. This type of approach follows a development that is very different from a direct and natural informal proof. For instance, the states of the target LTS of this encoding are processes enriched with additional information, so achieving the soundness results requires new up-to-techniques to deal with this new state. Our approach is considerably more conservative; we stay closer to the standard informal development of up-to-techniques for the π-calculus.

As we stated in Section 1, showing that two process calculus expressions are bisimilar typically involves demonstrating two facts: (1) that those process expressions are within a particular bisimulation-up-to relation and (2) that such a bisimulation-up-to relation is, in fact, sound. In this paper, we have focused only on this second problem. The first problem is more related to model checking than sophisticated interactive theorem proving. The Bedwyr model checker [2, 29] has been used to perform state exploration over labelled transition systems involving process calculi such as CCS and the π-calculus. Since the logic $\mathcal{G}$ underlying Abella also provides a formal foundation for the logic underlying Bedwyr, theorems proved by the latter can be accepted directly by the former. In this way, Abella and Bedwyr can be used together to provide a trusted basis for establishing bisimulation results.

7 Conclusions

Since the proof theory literature has now adequately addressed the necessary features of relational programming, least and greatest fixpoints, and binding structures in terms, and since the Abella theorem prover directly supports those innovations, Abella seems to be a natural setting to formalize the metatheory of bisimulation-up-to. We have shown this to be the case in this paper. In particular, we have shown that (1) the mathematical definitions of important relations could be written directly within the logic of Abella; (2) the available informal

theorems and proofs could be used to provide detailed outlines for the fully formal proofs; (3) we could lift in a straightforward fashion our definitions, theorems, and proofs involving a system without bindings to a system with bindings; and (4) we believe that we have written our formalism in a modular fashion that will allow us to attack the metatheory of a number of other up-to-techniques.

Acknowledgements This work was funded, in part, by the ERC Advanced Grant ProofCert and by the Inria Associated Team grant RAPT. We thank the anonymous reviewers for their comments on an earlier draft of this paper.

References

[1] D. Baelde, K. Chaudhuri, A. Gacek, D. Miller, G. Nadathur, A. Tiu, and Y. Wang. Abella: A system for reasoning about relational specifications. *Journal of Formalized Reasoning*, 2014. To appear.

[2] D. Baelde, A. Gacek, D. Miller, G. Nadathur, and A. Tiu. The Bedwyr system for model checking over syntactic expressions. In F. Pfenning, editor, *21th Conf. on Automated Deduction (CADE)*, number 4603 in Lecture Notes in Artificial Intelligence, pages 391–397, New York, 2007. Springer.

[3] J. Bengtson and J. Parrow. Formalising the π-calculus using nominal logic. *Logical Methods in Computer Science*, 5(2):1–36, 2009.

[4] F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 457–468. ACM, 2013.

[5] K. Chaudhuri, M. Cimini, and D. Miller. The Abella formalization of bisimulation-up-to for CCS and the π-calculus. Available from `http://abella-prover.org/upto`.

[6] J. Endrullis, D. Hendriks, and M. Bodin. Circular coinduction in Coq using bisimulation-up-to techniques. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP*, volume 7998 of *Lecture Notes in Computer Science*, pages 354–369, Rennes, July 2013. Springer.

[7] M. J. Gabbay. The π-calculus in FM. In F. D. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, volume 28 of *Applied Logic Series*, pages 80–149. Kluwer, 2004.

[8] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.

[9] A. Gacek. *A Framework for Specifying, Prototyping, and Reasoning about Computational Systems*. PhD thesis, University of Minnesota, 2009.

[10] A. Gacek, D. Miller, and G. Nadathur. Nominal abstraction. *Information and Computation*, 209(1):48–73, 2011.

[11] G. Gentzen. New version of the consistency proof for elementary number theory. In M. E. Szabo, editor, *Collected Papers of Gerhard Gentzen*, pages 252–286. North-Holland, Amsterdam, 1969. Originally published 1938.

[12] F. Honsell, M. Miculan, and I. Scagnetto. π-calculus in (co)inductive type theories. *Theoretical Computer Science*, 2(253):239–285, 2001.

[13] C.-K. Hur, G. Neis, D. Dreyer, and V. Vafeiadis. The power of parameterization in coinductive proof. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 193–206. ACM, 2013.

[14] J. Madiot, D. Pous, and D. Sangiorgi. Bisimulations up-to: Beyond first-order transition systems. In P. Baldan and D. Gorla, editors, *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR)*, volume 8704 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2014.

[15] D. Miller and G. Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, June 2012.

[16] D. Miller and C. Palamidessi. Foundational aspects of syntax. *ACM Computing Surveys*, 31, Sept. 1999.

[17] D. Miller and A. Tiu. A proof theory for generic judgments: An extended abstract. In P. Kolaitis, editor, *18th Symp. on Logic in Computer Science*, pages 118–127. IEEE, June 2003.

[18] D. Miller and A. Tiu. A proof theory for generic judgments. *ACM Trans. on Computational Logic*, 6(4):749–783, Oct. 2005.

[19] R. Milner. *Communication and Concurrency*. Prentice-Hall International, 1989.

[20] D. Pous. Weak bisimulation upto elaboration. In C. Baier and H. Hermanns, editors, *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 390–405. Springer, 2006.

[21] D. Pous. Complete lattices and upto techniques. In Z. Shao, editor, *APLAS*, volume 4807 of *Lecture Notes in Computer Science*, pages 351–366, Singapore, Nov. 2007. Springer.

[22] D. Pous and D. Sangiorgi. Enhancements of the bisimulation proof method. In D. Sangiorgi and J. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, pages 233–289. Cambridge University Press, 2011.

[23] D. Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998.

[24] D. Sangiorgi and R. Milner. The problem of "weak bisimulation up to". In *CONCUR*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1992.

[25] D. Sangiorgi and D. Walker. *π-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[26] A. Tiu and D. Miller. A proof search specification of the π-calculus. In *3rd Workshop on the Foundations of Global Ubiquitous Computing*, volume 138 of *ENTCS*, pages 79–101, Sept. 2004.

[27] A. Tiu and D. Miller. Proof search specifications of bisimulation and modal logics for the π-calculus. *ACM Trans. on Computational Logic*, 11(2), 2010.

[28] The Abella web-site. `http://abella-prover.org/`, 2015.

[29] The Bedwyr model checker, 2015. Available at `http://slimmer.gforge.inria.fr/bedwyr/`.