Coinduction All the Way Up
Damien Pous

To cite this version:
Damien Pous. Coinduction All the Way Up. Thirty-First Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2016), Jul 2016, New York, United States. 10.1145/2933575.2934564 . hal-01259622v2

HAL Id: hal-01259622
https://hal.archives-ouvertes.fr/hal-01259622v2
Submitted on 19 Jan 2017

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.
Coinduction All the Way Up

Damien Pous *
Université de Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP
Damien.Pous@ens-lyon.fr

Abstract

We revisit coinductive proof principles from a lattice theoretic point of view. By associating to any monotone function a function which we call the companion, we give a new presentation of both Knaster-Tarski’s seminal result, and of the more recent theory of enhancements of the coinductive proof method (up-to techniques).

The resulting theory encompasses parameterized coinduction, as recently proposed by Hur et al., and second-order reasoning, i.e., the ability to reason coinductively about the enhancements themselves. It moreover resolves a historical peculiarity about up-to context techniques.

Based on these results, we present an open-ended proof system allowing one to perform proofs on-the-fly and to neatly separate inductive and coinductive phases.

Categories and Subject Descriptors F.3 [Logics and meanings of programs]; F.4 [Mathematical logic and formal languages]

Keywords Coinduction, Enhancements, Complete lattices, GSOS, Parameterized coinduction

1. Introduction

Coinduction is a simple mathematical tool that follows from Knaster-Tarski’s fixpoint theorem about complete lattices [14, 33]. It was first used implicitly, for instance in finite automata algorithms [9, 10], until Milner popularised it by proposing bisimilarity as a natural way to compare concurrent programs [19]. It was then widely used, for instance to analyse other process calculi [20], the lambda-calculus [2], cryptographic protocols [1], distributed implementations [8], concurrent ML [13], analytic differential equations [27], or C compilers [17, 30].

The reason for such a success is that like induction, coinduction provides a powerful proof technique: to prove some property by coinduction, it suffices to exhibit an invariant. Typically, in program semantics, one can prove the equivalence of two programs by exhibiting a bisimulation relation that contains those two programs. The point there is that while program equivalence is a global property (for instance, because equivalent programs should remain equivalent under arbitrary contexts), the conditions for a relation to be a bisimulation are local. By using coinduction one can thus ensure a global property by checking only local properties. Very much like induction allows one to reduce a proof about arbitrary natural numbers to a proof about zero and successor.

From the beginning [19], Milner introduced enhancements of the bisimulation proof method. They make it possible to work with relations that are much smaller than actual bisimulations and yet ensure program equivalence: they are always contained in a bisimulation. Those relations are usually called bisimulations up to. The benefits of these enhancements can be spectacular; a bisimulation up to can be finite whereas any enclosing bisimulation is infinite. Sometimes it may be hard even to define an enclosing bisimulation, let alone carrying out the whole proof. There are many possible enhancements, and they proved useful, if not essential, in proofs about name-passing languages [6, 12, 29], languages with information hiding mechanisms (e.g., existential types, encryption and decryption constructs [1, 31, 32]), and higher-order languages [15, 16].

Sangiorgi developed a first theory of those enhancements [28, 29], which the present author further refined [23–25]. In this line of work, the emphasis was put on compositionality: given the wide variety of enhancements, it is crucial to have tools to analyse each of them separately, and then to combine them when needed in a concrete proof. Since enhancements do not compose in general, Sangiorgi proposed a notion of respectful enhancement. These form a subclass of the valid enhancements, and they enjoy nice compositional properties: they are closed under union and composition. One can thus establish a dictionary of respectful enhancements, and then use any combination of those in concrete proofs. In the author’s refinement of this framework, respectfulness was modified into compatibility, a slightly more natural notion which essentially plays the same role but leads to a smoother theory.

Recently, Hur et al. proposed parameterized coinduction [11], an extremely neat variation on Knaster-Tarski theorem which allows one to present coinductive proofs incrementally, without having to exhibit the invariant (or bisimulation relation) from the beginning. Such a possibility is especially useful in the context of mechanised formal proofs: the process of discovering the appropriate invariant becomes interactive and amenable to automation.

They also show in this paper how to exploit respectful enhancements with parameterized coinduction. When doing so, they define the greatest respectful enhancement and they remark in passing that it is so powerful that there is no point in using a different one.

In the present work, we start from this simple remark and we push it to the limit. Rather than studying respectful or compatible enhancements, we focus on the greatest one from the very beginning. Unexpectedly, doing so leads us to a greatly simplified abstract theory of enhanced coinduction, which encompasses respectfulness, compatibility, parameterized coinduction, and second-order reasoning. More precisely:

- We obtain an alternative proof of Knaster-Tarski’s theorem, where compatible enhancements play a central role. That com-
patible enhancements can safely be used then becomes a simple observation (Section 3).

• We characterise the greatest compatible enhancement as a greatest fixpoint in the complete lattice of monotone functions. This allows us to reuse our theory of enhanced coinduction at the second-order level, and to obtain powerful proof techniques for establishing the validity of enhancements. While we already used this approach before [23], focusing on the greatest compatible enhancement brings considerable simplifications and a much cleaner presentation (Section 6).

• Similarly, the formal development of symmetry arguments, which was rather painful to obtain before [23], gets greatly simplified (Section 7).

• The distinction between respectful and compatible enhancements vanishes in the new theory (Section 9).

• Parameterized coinduction, as proposed by Hur et al. becomes a • The first-order theory by considering bisimulation proofs in CCS (Section 4) and in Rutten’s stream calculus (Section 5). We moreover use the second-order theory to recover up-to-context and congruence results in a compositional and elementary way, for both CCS and the ∅-calculus (Section 8).

The presented theory and some of the examples have been formalised as a Coq library, which is available from the following webpage: http://perso.ens-lyon.fr/damien.pous/cawu

2. Notation and preliminary material

A complete lattice is a triple (X, ≤, V) where (X, ≤) is a partial order (reflexive, transitive, and antisymmetric) such that any subset Y of X has a least upper bound ∨Y: for all z ∈ X,

\[ \forall Y \subseteq z \iff \forall y \in Y, y \leq z \]

A complete lattice always has a bottom element, written ⊥, and a binary join operation, written with infix symbol ∨:

\[ \perp \equiv \bigvee \emptyset \quad \quad \quad x \lor y \equiv \bigvee \{x, y\} \]

Arbitrary greatest lower bounds can be derived from least upper bounds. We denote binary ones (meets) with the infix symbol ∧.

Standard examples of complete lattices include: subsets (of a given set) ordered with inclusion; binary relations (on a given set) ordered with inclusion again, and functions into a complete lattice, ordered pointwise. A fourth example, used thoroughly in this paper, is the set of monotone functions on a complete lattice. More precisely, given a complete lattice (X, ≤, V), a function f : X → X is monotone if it preserves the partial order:

\[ \forall x, y \in X, x \leq y \Rightarrow f(x) \leq f(y) \]

We write [X → X] for the set of monotone functions on X. When ordered pointwise, this set forms a complete lattice: for all f, g : [X → X] and F ⊆ [X → X],

\[ f \leq g \equiv \forall x \in X, f(x) \leq g(x) \]

\[ \bigvee f \equiv x \mapsto \bigvee_{f \in F} f(x) \]

A post-fixpoint of a function f : [X → X] is an element x such that x ≤ f(x); a fixpoint is an element x such that x = f(x).

In the abstract developments of this paper, we mostly work within a generic complete lattice X, the corresponding lattice of monotone functions [X → X], and that of monotone functions on [X → X]: [[X → X] → [X → X]]. To avoid confusion, we use the following convention: letters x, y, z range over elements of X, letters f, g, b, e, t range over functions in [X → X], and uppercase letters F, B, S, T are reserved for functions in [[X → X] → [X → X]]. We follow the same convention in concrete examples, except that we use bold fonts.

This discipline allows us to overload most symbols in the sequel: for instance, depending on the context, ⊥ can denote the empty set, the bottom element of an abstract complete lattice X, or the everywhere-bottom function in [X → X].

To further alleviate notation, we denote the identity function by 1, and both function composition and function application by juxtaposition:

• f(x) denotes the application of a function f to an element x, usually written f(x);

• \( fg \) denotes the composition of two functions f and g, usually written \( f \circ g \).

(Similarly for \( Fg \) and \( FB \).) We associate juxtapositions to the right when there is no ambiguity. For instance, we write \( f \cdot x \) for \( f(x) \cdot x \), \( F \cdot g(x) \cdot x \) for \( f(g(x)) \cdot x \), and \( T \cdot T \cdot f \) for \( T \cdot (T \cdot f) \). In contrast, we keep parentheses in expressions such that \( (Bf) \cdot g \) and \( B \cdot (f \cdot g) \) which are not equal in general.

3. Knaster-Tarski and Compatibility

We fix a complete lattice (X, ≤, V) and pick a monotone function \( b : [X \rightarrow X] \) throughout the paper. Knaster-Tarski’s theorem characterises the greatest fixpoint \( vb \) of \( b \) as the least upper bound of all its post-fixpoints:

\[ vb = \bigvee_{x \leq vb} x \]

The corresponding coinduction principle is given on the right-hand side. In words, to prove that \( x \) is below in the greatest fixpoint, find a post-fixpoint \( y \) above \( x \). The idea of enhancements is to use an additional function \( f \) and to look for post-fixpoints of \( fb \) rather than \( b \); we switch to the following principle of coinduction up to \( f \)

\[ x \leq y \leq fb \Rightarrow bx \leq vb \]

(2)

The function \( f \) typically enlarges its argument, and the post-fixpoints of \( fb \) can be much smaller than those of \( b \); these are the bisimulations up to we alluded to in the Introduction. The function \( f \) corresponds to a valid enhancement when the above rule holds, or, equivalently, when \( vb \) \( fb \) \( vb \). Our primary goal is to obtain such functions.

A monotone function \( f : [X \rightarrow X] \) is compatible (for \( b \)) if \( fb \leq fb \). It is straightforward to check that 1 and \( b \) are compatible, that the composition of two compatible functions is compatible, and that the least upper bound of a family of compatible functions is compatible.

**Definition 3.1.** We call companion of \( b \) the monotone function obtained as the least upper bound of all compatible functions:

\[ t \equiv \bigvee_{fb \leq fb} f \]

**Lemma 3.2.** The companion is compatible:

\[ tb \leq bt \]
Thus this is the greatest compatible function. It moreover satisfies

\[ b \leq t \]  
\[ 1 \leq t \]  
\[ tt \leq t \]  

The last two inequalities entail idempotence, i.e., \( tt = t \).

**Proof.** Inequality (3) follows from the fact that supremum of a family of compatible functions is compatible. That it contains \( b \) and the identity (4,5) follows from the fact that those functions are compatible. For (6), it suffices to notice that \( tt \) is compatible, by (3) and the fact that composition preserves compatibility. \( \square \)

Our first result is the following alternative definition of the greatest (post-)fixpoint, using the companion:

**Theorem 3.3.** The greatest fixpoint of \( b \) is the value of the companion on the bottom element.

\[ \nu b = t \perp \]  

**Proof.** We first show that \( t \perp \) is the greatest post-fixpoint:

1. \( t \perp \) is a post-fixpoint: we have \( t \perp \leq b \perp \leq b t \perp \) by monotonicity and compatibility of \( t \);
2. it is the largest: if \( x \leq b x \), then the constant-to-\( x \) function \( \overline{x} \) is compatible and thus smaller than \( t \), so that \( x = b \perp \leq t \perp \).

We conclude that \( t \perp \) is a fixpoint as in Knaster-Tarski’s proof: from monotonicity of \( b \) and the first point, \( b \perp \perp \) is also a post-fixpoint, and thus \( b t \perp \leq t \perp \) by the second point. \( \square \)

Using idempotence of the companion, we also get

**Corollary 3.4.** The companion preserves the greatest fixpoint:

\[ tvb = vb \]  

Typically, when \( b \) is the function defining bisimilarity on some process calculus, we recover the fact that if contextual closure is compatible (and thus below \( t \)) then bisimilarity is closed under contexts.

**Definition 3.5.** We write \( b^\dagger \) for the composite function \( b t \).

This function is an improved version of \( b \), with more post-fixpoints (we have \( b \leq b^\dagger \)) but the same greatest fixpoint:

**Theorem 3.6.** The companion is a valid enhancement, we have

\[ vb^\dagger = vb \]  

**Proof.** From (5) we deduce \( b \leq b^\dagger \), and thus \( vb \leq vb^\dagger \). The interesting result is the other inequality. Since \( vb^\dagger \) is the greatest post-fixpoint of \( b^\dagger \), it suffices to show that any post-fixpoint \( x \) of \( b^\dagger \) is smaller than \( vb \). We have

\[ tx \leq tb^\dagger x = bttx \quad \text{ (assumption on } x \text{ and monotonicity of } t) \leq bttx \quad \text{ (it is compatible (3))} \leq bttx = b^\dagger x \quad \text{ (by (6) and monotonicity of } b) \]

Thus \( tx \) is a post-fixpoint of \( b \), and \( tx \leq vb \). We conclude with (5): we have \( x \leq tx \leq vb \). \( \square \)

**Remark 3.7.** In earlier work by Sangiorgi [28] and then by the author [23], where the emphasis was on compatible functions rather than on the greatest one (the companion), the corresponding result is “if \( f \) is compatible, and \( x \leq bf x \) then \( x \leq vb^\dagger f \). Such a result requires a convoluted proof. Indeed, when \( f \) is an arbitrary compatible function, one does not have \( 1 \leq f \) and \( f f \leq f \), and the above proof breaks. Instead, one constructs the sequence

\[ A \triangleq P \quad P \xrightarrow{\alpha} P' \quad \alpha. P \xrightarrow{\alpha} P \]

\[ P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\alpha} Q' \quad P|Q \xrightarrow{\alpha} P'|Q' \]

**Figure 1.** Labelled transition system of a fragment of CCS.

\[ f^0 x \triangleq x \quad f^{i+1} x \triangleq f f^i x \quad x \]  

One might hope to enhance the function \( b \) further by using the companion of \( b^\dagger \). However we stagnate when doing so: let \( t^\star \) be the companion of \( b^\dagger \), we have

\[ t^\star = t \]  

(10)

\[ b^\dagger = b^\dagger \]  

(11)

(The proof is given in appendix; note that Equation (10) actually generalises Theorem 3.6: we have \( vb^\dagger = t \perp = t \perp = vb \).)

4. **Modularity through the companion**

We consider a first example in a fragment of Milner’s CCS [19]. Let us recall this process calculus first. We fix a set of names \( a, b, \ldots \), and a set of process constants \( A, B, \ldots \). CCS processes and *labels* are defined by the following grammar:

\[ P, Q ::= A \mid 0 \mid a.P \mid P|P \mid a.P \mid \pi \mid \tau \]

We let \( R, S \) range over binary relations on processes.

The corresponding labelled transition system (LTS) is given in Figure 1. The first rule accounts for recursion: it assumes that each process constant is associated to a process in some global table. The two symmetrical rule for parallel composition are omitted.

Let \( b \) be the following monotone function on the lattice of binary relations on processes:

\[ b : R \mapsto \{ (P, Q) \mid \forall \alpha, \forall P', \exists Q', Q \xrightarrow{\alpha} Q' \text{ and } P' \not\sim R Q' \} \]

\[ \forall Q', Q \xrightarrow{\alpha} Q' \text{ entails } 3Q', P \xrightarrow{\alpha} P' \text{ and } P' \not\sim R Q' \}

The so-called bisimulations are the post-fixpoints of \( b \), and bisimilarity (\( \sim \)) is its greatest-fixpoint.

**An enhanced coinductive proof.** Consider the following process definitions, and let us try to prove that \( A \sim B \).

\[ A \triangleq a.b.D \quad B \triangleq a.b.C \]

\[ C \triangleq \pi.(A(C)) \quad D \triangleq \pi.(B(D)) \]

Any bisimulation containing the pair \( \langle A, B \rangle \) must be infinite. Instead, the companion of \( b \) allows us work with the finite relation \( S \triangleq \{ \langle A, B \rangle, \langle C, D \rangle \} \); we have \( S \leq b^\dagger S \).

Indeed, we have \( A \xrightarrow{\alpha} b.D \) and \( B \xrightarrow{\alpha} b.C \). While the pair \( b.D, b.C \) does not belong to \( S \), we can use the following functions to cancel the \( b \) prefixes and to transpose \( C \) and \( D \):

\[ e : R \mapsto \{ (a.P, a.Q) \mid a \text{ a label, } P \not\sim R Q \} \]

\[ i : R \mapsto \{ (Q, P) \mid P \not\sim R Q \} \]

We thus have \( \langle A, B \rangle \in bie S \). The function \( i \) is trivially compatible for \( b \), and we shall see in Section 8.1 that \( e \) is below the
companion of b, written t in the sequel. Whence \( \text{ic} \leq t \leq t \), and thus \( \langle A, B \rangle \in b^1S \).

Similarly, we have \( C \xrightarrow{\sigma} A | C \) and \( D \xrightarrow{\tau} B | D \), and we use the following function to cancel parallel composition and recover the two pairs from \( S \):

\[
c^1 : \mathcal{R} \rightarrow \{ (P, P'), Q, Q' \mid P \mathcal{R} Q, P' \mathcal{R} Q' \}
\]

This function is below t (see Section 8.1 again), so that we get \( (C, D) \in b^1c^1S \leq b^1S \).

Note that thanks to the companion, we only had to study transitions along labels \( \alpha \) and \( \bar{\alpha} \), even though the processes at hand also perform transitions labelled \( \beta \) and \( \bar{\beta} \). (For instance, we have \( A \xrightarrow{\sigma \beta \tau} b.C | B | D \).) The fact that the starting processes cannot diverge one from the other using those actions is somehow factored once and for all, in the proofs that \( c^1 \) and \( c^1 \) are valid enhancements.

**Modularity.** As pointed out by Hur et al. [11], working with the companion rather than with specific compatible functions is quite convenient: it does not require us to announce a global up-to-technique up-front. (Here, something like companion rather than with specific compatible functions is quite needed for that case. This approach is much more robust, especially in the context of computer-assisted proofs. Suppose for instance that one slightly changes the definition of \( D \) into \( b.(D | B) \). One can still conclude by reasoning up to commutativity of parallel composition, and this additional technique is already available in the companion: there is no need to update the declared up-to-technique, one just needs to adjust the proof locally. (Of course one needs to prove that this new kind of enhancement is available in the companion, but this can be done separately, and once and for all.)

**Code reuse.** Although this was not needed in the previous example, one can also show that the following function is compatible:

\[
j : \mathcal{R} \rightarrow \{ (P, R) | \exists Q, P \mathcal{R} Q, Q \mathcal{R} R \}
\]

Thus \( j \leq t \), and together with (6), \( j \mathcal{R} t \). In other words, \( t \mathcal{R} \) is transitive for any relation \( \mathcal{R} \). More generally, from \( c^1, c^1, i, j \leq t \) and \( tt \leq t \), we deduce that for any relation \( \mathcal{R} \), \( t \mathcal{R} \) is a congruence containing both \( \mathcal{R} \) and \( \sim \).

In the context of proof assistants, this simple realisation makes it possible to reuse standard technology for automating equational reasoning (e.g., in the Coq proof assistant, setoid rewriting).

Also note that since \( \sim \approx t \mathcal{R} \), we obtain as a special case that bisimilarity is a congruence. In particular, once the aforementioned technology has been settled for \( t \mathcal{R} \) for an arbitrary \( \mathcal{R} \), tools for equational reasoning about bisimilarity come for free.

5. **Streams**

As a second example, we consider the *stream calculus*, as developed by Rutten [27]. Let us denote by \( \mathbb{R}^\omega \) the set of streams, i.e., infinite sequences \( \sigma \), \( \tau \), . . . of real numbers.

Together with the following function associating to each stream its first element and its tail, \( \mathbb{R}^\omega \) is a final coalgebra for the functor \( F \mathbb{R} = \mathbb{R} \times X \)

\[
\mathbb{R}^\omega \rightarrow \mathbb{R} \times \mathbb{R}^\omega
\]

\[
\sigma \mapsto (\sigma_0, \sigma')
\]

One can thus define streams by *behavioural differential equations* (i.e., \( F \)-coalgebras). For instance, the everywhere-0 stream 0 can be defined by the following equations:

\[
\bar{0}_0 = 0 \quad \bar{0}' = 0
\]

Similarly, pointwise addition of streams can be defined by

\[
(\sigma + \tau)_0 = \sigma_0 + \tau_0 \quad (\sigma + \tau)' = \sigma' + \tau'
\]

**Shuffle product.** Things become more interesting for more complex operations. Take for instance the *shuffle product* of streams, usually defined by the following formula:

\[
(\sigma \shuffle \tau)_n = \sum_{k=0}^{\infty} \left( \binom{n}{k} \times \sigma_k \times \tau_{n-k} \right)
\]

This operation can alternatively be defined using the following differential equations, which no longer involve binomial coefficients:

\[
(\sigma \shuffle \tau)_0 = \sigma_0 \times \tau_0 \quad (\sigma \shuffle \tau)' = \sigma' \times \tau + \sigma \times \tau'
\]

As noticed by Rutten, proving a simple property like associativity can be difficult with the former definition, as it would involve double summations of terms with several binomial coefficients. In contrast, one can give a straightforward inductive proof.

Let \( b \) be the following (monotone) function on binary relations on streams:

\[
b : \mathcal{R} \rightarrow \{ (\sigma, \tau) | \sigma_0 = \tau_0 \text{ and } \sigma' \mathcal{R} \tau' \}
\]

One easily checks that its greatest fixpoint is just the identity relation: \( (\sigma, \tau) \in \mathcal{R} b \) if \( \sigma = \tau \). One can thus prove stream equalities by coinduction.

As a trivial example consider commutativity of stream addition: it is immediate to see that the relation \( \{ (\sigma + \tau, \tau + \sigma) | \sigma, \tau \in \mathbb{R}^\omega \} \) is a post-fixpoint of \( b \); this relation is thus contained in the identity, and stream addition is commutative.

Coming back to associativity of the shuffle product, we might accordingly try to use the following relation:

\[
S \triangleq \{ ((\sigma \shuffle \tau) \shuffle \rho, \sigma \shuffle (\tau \shuffle \rho)) | \sigma, \tau, \rho \in \mathbb{R}^\omega \}
\]

Unfortunately, this relation is not a post-fixpoint of \( b \): assuming distributivity has already been proved, we have

\[
((\sigma \shuffle \tau) \shuffle \rho)' = (\sigma \shuffle (\tau \shuffle \rho))' = (\sigma \shuffle \tau) \shuffle (\tau \shuffle \rho) + (\sigma \shuffle (\tau \shuffle \rho))
\]

and those two streams are not related by \( S \). Like in the previous example in CCS, we would like to cancel the two sums on both sides, in order to recover three pairs in \( S \). This is possible using the companion of \( b \); the following function is easily shown to be compatible for \( b \), so that \( S \leq b^1S \).

\[
c^1 : \mathcal{R} \rightarrow \{ (\sigma + \rho, \tau + \omega) | \sigma \mathcal{R} \tau, \rho \mathcal{R} \omega \}
\]

We have thus obtained a straightforward proof of associativity of the shuffle product.

**Exponentiation.** Let us finally consider a third natural operation on streams: *exponentiation*, defined by the following differential equation:

\[
e^\sigma_0 = e^\sigma_0 \quad e^{\sigma + \tau} = e^\sigma \otimes e^\tau
\]

As expected, we have \( e^{\sigma + \tau} = e^\sigma \otimes e^\tau \). To prove it by following the same path as above, one needs to cancel a shuffle product, thus calling for the following function:

\[
e^{\sigma} \mathcal{R} \rightarrow \{ (\sigma \mathcal{R} \rho, \tau \mathcal{R} \omega) | \sigma \mathcal{R} \tau, \rho \mathcal{R} \omega \}
\]

While this function is indeed below the companion of \( b \), it is not compatible for \( b \).

To understand why, let us try to prove compatibility of this function, i.e., \( e^\sigma b \leq b e^\sigma \). Let \( \mathcal{R} \) be a relation. We have

\[
e^{\sigma} b \mathcal{R} = \{ (\sigma \mathcal{R} \rho, \tau \mathcal{R} \omega) | \sigma \mathcal{R} \tau, \rho \mathcal{R} \omega \}
\]

So assuming \( \sigma \mathcal{R} \tau \) and \( \rho \mathcal{R} \omega \), we have to show that \( (\sigma \mathcal{R} \rho, \tau \mathcal{R} \omega) \) is easy:...
the problem comes from the tails of those streams:

\[(\sigma \otimes \rho)^\prime \leq \sigma^\prime \otimes \rho + \sigma \otimes \rho^\prime\]

\[(\tau \otimes \omega)^\prime \leq \tau^\prime \otimes \omega + \tau \otimes \omega^\prime\]

First we need to cancel the sum operation (using \(e^+\)). Second, while we have \(\sigma^\prime \mathcal{R} \tau^\prime\) and \(\rho^\prime \mathcal{R} \omega^\prime\) by assumption, we only have \(\rho \mathcal{R} \omega\) and \(\sigma \mathcal{R} \tau\). Instead, in the end, instead of \(e^+ b \leq e^\circ (b \mathcal{V} 1)\), we have

\[e^\circ b \leq b e^+ (b \mathcal{V} 1)\]

(12)

We shall see in the following section that such a result nevertheless ensures that the function \(e^\circ\) is below the companion of \(b\), and can thus safely be used in enhanced coinductive proofs about streams.

6. Compatibility up-to

In this section we show that the companion is a coinductive object. This gives us powerful proof techniques to obtain enhancements.

**Definition 6.1.** Let \(B\) be the following function from monotone functions on \(X\) to monotone functions on \(X\):

\[B : [X \to X] \to [X \to X]\]

\[g \mapsto \bigvee_{f b \leq g} f\]

**Lemma 6.2.** \(B\) is monotone, and for all functions \(f, g : [X \to X]\),

\[f \leq Bg \iff fb \leq bg\]

(13)

In particular, \(f\) is compatible if and only if it is a post-fixpoint of \(B\), so that the companion is the greatest fixpoint of \(B\):

\[t = \nu B\]

(14)

We can thus reuse the machinery from Section 3 with \(B\), in the second-order lattice \([X \to X]\). Let \(T\) be the companion of \(B\), and let \(B^\uparrow \triangleq BT\).

In the previous section, in the example about streams, we had a first function \(e^+\), which was compatible and thus trivially below the companion. In contrast, we claimed that the function \(e^\circ\) was below the companion, although it is not compatible: we do not have \(e^\circ b \leq b e^+,\) i.e., \(e^\circ \leq e^+\) (where \(B\) is the higher-order function associated to the function \(b\) by Definition 6.1). Instead, we had \(e^\circ \leq B(e^+ e^\circ (b \mathcal{V} 1))\). Using the results below, we will deduce from this inequality that \(e^\circ \leq B^\uparrow e^\circ\), so that \(e^\circ \leq \nu B\): the function \(e^\circ\) indeed lies below the companion.

In a sense, we face the standard scenario of bisimulation proofs, but in the lattice of monotone functions: \(e^\circ\) is an obvious coinductive candidate, but it is too weak, we should strengthen it to get a post-fixpoint. Luckily, instead of doing so, we can use an enhancement to build on the knowledge accumulated so far about the companion (in this case, that it contains \(e^+\), amongst other things).

Getting back to the abstract framework, the second-order companion \(T\) enjoys many good properties, listed in Proposition 6.4 below. We first need to establish a compatibility result for \(B\).

**Lemma 6.3.** The function \(S : f \mapsto ff\) is compatible for \(B\).

**Proof.** We have

\[SB \leq BS\]

iff \[\forall f, (Bf)(Bf) \leq B(ff)\] (by definition of \(S\))

iff \[\forall f, (Bf)(Bf)b \leq bf\] (by (13))

Fix some monotone function \(f : [X \to X]\). By taking \(g = f\) and \(f = Bf\) in (13) we have \((Bf)b \leq bf\). Thus we get

\[(Bf)(Bf)b \leq (Bf)bf \leq bf\]

and \(S\) is compatible for \(B\).

\[\square\]

**Proposition 6.4.** For any function \(f : [X \to X]\), we have

\[t \leq Tf\] (15)

\[b \leq Tf\] (16)

\[1 \leq Tf\] (17)

\[f \leq Tf\] (18)

\[TTf \leq Tf\] (19)

\[(Tf)(Tf) \leq Tf\] (20)

In particular, \(Tf\) is always an idempotent function.

**Proof.**

• (15): recall that \(t = T1\), and \(T\) is monotone;

• (16),(17): trivial consequences of the previous point and (4) and (5), respectively;

• (18),(19): respective instances of (5) and (6) for \(B\) (thus moving to the lattice \([X \to X]\));

• (20): by Lemma 6.3 we get \(S \leq T\). Then it suffices to compute:

\[(Tf)(Tf) = STf \leq TTf \leq Tf\]

\[\square\]

Note that the way we obtain (20) is very similar to the way we establish transitivity of \(\mathcal{R}\) at the end of Section 4.

Coming back to the example about the shuffle product on streams, write \(t\) and \(T\) for the companions of \(b\) and \(B\). It is now straightforward to check that \(e^+ e^\circ (b \mathcal{V} 1) \leq Te^\circ\):

\[e^+ e^\circ (b \mathcal{V} 1) \leq te^\circ (b \mathcal{V} 1)\]

\[\leq (Te^\circ)e^\circ (Te^\circ \mathcal{V} Te^\circ)\]

(by (15), (16), and (17))

\[\leq (Te^\circ)(Te^\circ)(Te^\circ)\]

(by (18))

\[\leq Te^\circ\]

(by (20) twice)

So from (12) we deduce \(e^\circ \leq B^\uparrow e^\circ\) and thus \(e^\circ \leq t\), as announced earlier.

(Note that we chose to make the function \(e^+ e^\circ (b \mathcal{V} 1)\) explicit here for the sake of explanation. In a direct proof, one would prove \(e^\circ \leq B^\uparrow e^\circ\) by extracting the required components out of \(Te^\circ\) on the fly, exactly as we did in Section 4 but at the second-order level.)

7. Symmetry arguments

We now give a rather general result allowing to exploit symmetry arguments in various proofs. This result formally justifies standard practice in bisimulation proofs with paper and pencil. When it comes to formal, mechanised, proofs, it is crucial to have such results, to factor the code and avoid cut-and-paste.

Let \(i : [X \to X]\) be a monotone involution on \(X\):

\[ii = 1\]

(21)

Call an element \(x \in X\) symmetric if \(ix = x\) (which is equivalent to \(ix \leq x\) thanks to (21)). Call a function \(f : [X \to X]\) symmetric if \(fi = if\) (which is equivalent to \(f\) being compatible for \(i\), again using (21)).

As a concrete example in the lattice of binary relations, the natural candidate for the function \(i\) is the transposition function from Section 4:

\[i : \mathcal{R} \rightarrow \mathcal{R}^{-1} = \{\langle Q, P \rangle \mid P \mathcal{R} Q\}\]

With such a choice, a relation \(\mathcal{R}\) is symmetric if \(\mathcal{R}^{-1} = \mathcal{R}\), and a function \(f\) is symmetric if \(f(\mathcal{R}^{-1}) = f(\mathcal{R})^{-1}\) for all relation \(\mathcal{R}\).

When the function \(b\) is symmetric, the following proposition can be used to factor proofs about symmetric candidates, both at the level of elements (\(X\)) and at the level of enhancements (\([X \to X]\)). We instantiate this result in the following section, when reasoning about bisimilarity in CCS and the \(\pi\)-calculus.
With such a function, we only play from left to right. The post-fixpoints of $s$ are the simulations, and its greatest fixpoint is similarity. The composite function $isi$ corresponds to simulations again, but played from right to left. As expected the function $b$ for bisimilarity thus decomposes as required in Proposition 7.1:

$$b = s \land isi$$  \hspace{1cm} (24)

Applied in this setting, equivalence (22) is not so surprising: when analysing the transitions of a symmetric bisimulation candidate, we can restrict ourselves to the left-to-right part of the bisimulation game. Note that thanks to the companion, we do not need $y$ to be symmetric (because $ty$ is). The second equivalence (23) is quite important in the sequel: one can also restrict ourselves to the left-to-right part of the bisimulation game when analysing the behaviour of a potential enhancement, provided it is symmetric.

Following closely the syntax of the calculus, we define the following functions on binary relations:

- $e^1 : R \mapsto \{(a,P,\alpha.Q) \mid \alpha \text{ a label}, P \ R Q\}$
- $c^1 : R \mapsto \{(|P,P',Q,Q'| P \ R Q, P' \ R Q')\}$
- $e^+ : R \mapsto \{(P + P', Q + Q') P \ R Q, P' \ R Q'\}$
- $c^+ : R \mapsto \{(|P,P'| Q \ R Q)\}$
- $c^\cdot : R \mapsto \{(|\nu a,P,\nu a.Q) \mid \alpha \text{ a name}, P \ R Q\}$

The functions $e^1$ and $c^1$ have already been defined in Section 4; we include them here to emphasise the uniformity of those definitions.

Let $c$ be one of the above functions; we want to prove $c \leq t$, where $t$ is the companion of $b$. From the results of Section 6, it thus suffices to prove $cb \leq bTc$, where $T$ is the companion of the second-order function $B$ associated to $b$. And since all the above functions are symmetric, it suffices by (23) to prove $cb \leq sTc$.

For the “dynamic” operations that disappear after a single transition (functions $c^1$ and $c^+$), we actually do not need coinduction at all. Routine computations lead to

$$c^1 b \leq s$$

(For $c^1$, we have $c^1 \leq s$.) This is fine because $b,1 \leq T \bot (16,17)$.

Instead, we do need coinduction for the “static” operations, which persist through transitions. The simplest is name restriction: we have $c^\cdot b \leq sc^\cdot$, and $c^\cdot e^\cdot \leq Tc^\cdot$ by (18). Parallel composition and replication require more effort, but even with this, we give detailed proofs to better illustrate our method.

**Lemma 8.1.** We have $c^1 b \leq sTc^1$, whence $c^1 \leq t$.

*Proof.* Let $R$ be a relation, an let $P, R, Q, S$ be processes such that $(P, Q), (R, S) \in bR$. We have to show that $(P|\nu a,Q|S)$ belongs to $sTc^1$. Thus suppose that $P|R \overset{\alpha}{\Rightarrow} P_\beta$ and let us find some $Q_\beta$ such that $Q|S \overset{\alpha}{\Rightarrow} Q_\beta$ and $(P_\beta, Q_\beta) \in Tc^1R$. There are four cases according to the rules of parallel composition (Figure 1):

1. $P_\beta = P'|R'$ with $\alpha = \tau$, and for some name $a$, $P \overset{\alpha}{\Rightarrow} P'$, and $R \overset{\alpha}{\Rightarrow} R'$. From our assumptions about $(P, Q)$ and $(R, S)$, we obtain processes $Q'$ and $S'$ such that $Q \overset{\alpha}{\Rightarrow} Q'$, $S \overset{\alpha}{\Rightarrow} S'$, $P' \ R Q'$ and $R' \ R S'$. We deduce that $Q|S \overset{\alpha}{\Rightarrow} Q'|S'$, and the pair $(P'|R', Q'|S')$ belongs to $c^1R$ and hence $Tc^1R$ by (18).

2. same as above but with $a$ and $\beta$ exchanged.

3. $P_\beta = P'|R$ with $P \overset{\alpha}{\Rightarrow} P'$. From the hypothesis about the pair $(P, Q)$ we obtain a process $Q'$ such that $Q \overset{\alpha}{\Rightarrow} Q'$ and $P' \ R Q'$. We deduce that $Q|S \overset{\alpha}{\Rightarrow} Q'|S$, and it remains to show $$(P'|R, Q'|S) \in Tc^1R$$
This is not as direct as before: rather than $R \mathcal{R} S$, we have $R \mathcal{R} S$. From (18) and (20), we have $e^i \mathcal{T}^c \subseteq \mathcal{T}^c$. Therefore, it suffices to show that $P^{(i)} \mathcal{T}^c \mathcal{R} Q^{(i)}$ and $R \mathcal{T}^c \mathcal{R} S$. The former holds thanks to (17); for the latter we use (16) instead.

4. $P_0 = P|R' \mathcal{R} R'$. This case is handled as above.

(Note that the above proof amounts to proving $e^i \mathcal{b} \subseteq \mathcal{b} \mathcal{v} 1$ and then showing that $e^i (\mathcal{b} \mathcal{v} 1) \subseteq \mathcal{T}^c$ using the generic properties of $T$—Proposition 6.4.)

Finally consider replication. This operation is quite challenging as far as up-to techniques are concerned: there was a slight mistake in [29], it requires specific rule formats [26], and people formalising up-to techniques in proof assistants have eluded this operation so far [7, 22]. The techniques developed above allow us to give a much cleaner proof, thus amenable to formalisation.

Let us first discuss the LTS rule for replication.

- The one presented in Figure 2 is standard; it makes it trivial to prove $!P \sim!P|P$ (which is the intended meaning) but it has the drawback of making the LTS infinitary branching.
- The obvious rule $*P \xrightarrow{a} P'$ entails $!P \xrightarrow{a} !P'$ is not enough in presence of choice, as it does not allow $!(|a.0 + a.0)$ to perform internal transitions.
- In his book about the π-calculus [29], Sangiorgi prefers to inline the behaviour of parallel composition to keep the LTS finitely branching. But this leads to duplication: there are two rules for replication in CCS, and three in the π-calculus.

Here we propose a path which is indifferent about this choice: it relies only on the following proposition, which is always valid in CCS and in π, whatever their presentation.

**Proposition 8.2.**

(i) If $!P \xrightarrow{a} P_0$, then there exists $P_1$ such that $P|P \xrightarrow{a} P_1$ and $P_0 \sim!P|P_1$.

(ii) Conversely, if $P|P \xrightarrow{a} P_1$, then there exists $P_0$ such that $!P \xrightarrow{a} P_0$ and $P_0 \sim!P|P_1$.

We need another preliminary result:

**Lemma 8.3.** The following function is compatible.

$$k : \mathcal{R} \mapsto \{(P, Q) \mid \exists P', Q', P \sim P', P' \mathcal{R} Q', Q' \sim Q\}$$

**Lemma 8.4.** We have $e^i \mathcal{b} \subseteq \mathcal{T}^c$, whence $e^i \subseteq t$.

**Proof:** Let $\mathcal{R}$ be a relation, let $(P, Q) \in \mathcal{b} \mathcal{R}$. We have to show that $(P, Q)$ belongs to $\mathcal{T}^c \mathcal{R}$. Thus suppose that $!P \xrightarrow{a} P_0$ and let us find some $Q_0$ such that $!Q \xrightarrow{a} Q_0$ and $(P_0, Q_0) \in \mathcal{T}^c \mathcal{R}$.

By Proposition 8.2(i), there is some $P_1$ such that $P|P \xrightarrow{a} P_1$ and $P_0 \sim!P|P_1$. Now notice that

$$\langle P|P, Q|Q \rangle \in e^i \mathcal{b} \mathcal{R} \quad \text{(by definition of } e^i \mathcal{b}) \leq tb \mathcal{R} \quad \text{(by Lemma 8.1)} \leq bt \mathcal{R} \quad \text{(the companion is compatible (3))}$$

By definition of $\mathcal{b}$, we thus obtain a process $Q_1$ such that $!Q \xrightarrow{a} Q_1$ and $P_1 \mathcal{R} Q_1$. We continue with Proposition 8.2(ii) which gives us $Q_0$ such that $!Q \xrightarrow{a} Q_0$ and $Q_0 \sim!Q|Q_1$.

Summarising our assumptions, we have

$$!P \xrightarrow{a} P_0 \sim!P|P_1 \quad P \mathcal{b} \mathcal{R} Q \quad P_1 \mathcal{R} Q_1 \quad !Q|Q_1 \sim Q_0 \mathcal{b}|Q$$

It remains to show that $(P_0, Q_0) \in \mathcal{T}^c \mathcal{R}$. This follows from the closure properties of $\mathcal{T}$ (Proposition 6.4), and $e^i, k \leq t$ (Lemmas 8.1 and 8.3). □

![Figure 3. LTS rules for prefix, parallel composition, and name restriction in the π-calculus.](image)

### 8.2 The π-calculus

The π-calculus [20] differs from CCS in that names can be passed along synchronisations. We briefly recall the main definitions here, referring to Sangiorgi’s book for a more detailed exposition of this calculus [29]. The syntax of processes remains the same as in CCS, only the notion of prefix changes:

$$P, Q ::= A | 0 | \pi.P | P|P | P + P | (\nu a)P | !P$$

$$\pi ::= \langle a,b \rangle | \mathcal{b}n$$

Prefixes in the $\pi$-calculus.

The prefix $a(b)$ denotes the input on $a$ of a name $b$, bound in the continuation; $\mathcal{b}n$ denotes the output on $a$ of a name $b$. Labels include two additional constructs: $\tau$ for internal communications, and $\mathcal{b}n$, for the emission of a private name $b$, bound in the continuation.

Concerning the (early) LTS, the rules for process constants, choice, and replication are the same as in CCS, the other ones are given in Figure 3. The three symmetrical rules for parallel composition are omitted. Processes are considered modulo alpha equivalence. The functions $n(\cdot)$ (names of a label) and $\mathcal{b}n(\cdot)$ (bound names of a label) are defined as follows:

$$\begin{array}{|c|c|c|c|}
\hline
\alpha & n(\alpha) & \mathcal{b}n(\alpha) & \mathcal{b}n(\alpha) \\
\hline
\langle a,b \rangle & \{a,b\} & \emptyset & \emptyset \\
\tau & \emptyset & \{a,b\} & \emptyset \\
\hline
\end{array}$$

To define (early) bisimilarity, one can import the function $\mathcal{b}$ from CCS almost as is. The only peculiarity is that we should ignore bound output transitions $\mathcal{b}n$ when $b$ occurs free in the answering process. Indeed, one should equate the following processes

$$P \triangleq (\nu c)\mathcal{b}n.0 \quad \mathcal{b} \triangleq P||\nu (\nu c)\mathcal{b}n.0$$

(Because $(\nu c)\mathcal{b}n.0$ is clearly equivalent to 0.) However, while $P$ can perform a transition labelled $\mathcal{b}n$ for any $b \neq a$, the process $Q$ can only perform such a transition when $b \neq a$, $d$, because of its free (yet useless) occurrence of $d$.

Whence the appropriate definitions of functions $s$ and $\mathcal{b}$:

$$s : \mathcal{R} \mapsto \{(P, Q) \mid \forall \alpha, \mathcal{b}n(\alpha) \#Q \text{ and } P \xrightarrow{\alpha} P' \text{ entail } \exists Q', Q \xrightarrow{\alpha} Q' \text{ and } P' \mathcal{R} Q'\}$$

$$\mathcal{b} \triangleq s \land s i$$

The functions $e^i, e^c, e^c, e^i$ are defined exactly as in CCS; we however replace the function $e^i$ with the following two ones:

$$e^i : \mathcal{R} \mapsto \{(\mathcal{b}n.P, \mathcal{b}n.Q) \mid a, b \text{ names, } P \mathcal{R} Q\}$$

$$e^c : \mathcal{R} \mapsto \{(a(b).P, a(b).Q) \mid a, b \text{ names, } \forall c, P^{(c/\cdot)} \mathcal{R} Q^{(c/\cdot)}\}$$

The reason why we need such a quantification on all names $c$ in the definition of $e^i$ is that bisimilarity is not preserved by input prefixes in the π-calculus: it is not closed under substitution, and the input prefix brings such substitutions. Thus, the plain function
corresponding to input prefix cannot be below the companion (it always preserves bisimilarity (8))). Instead, the above definition of \( e' \) takes into account the fact that \( b \) might be substituted later on.

Like in CCS, it is routine to check that we have
\[
\begin{align*}
\forall c \leq t \text{ for } c \in \{ e^+, e^+, e^+ c \}. \quad \text{Name restriction is no longer compatible stricto-senso; we have}
\end{align*}
\]
\[
\begin{align*}
e^+ b \leq s b \quad & \quad |c|' b \leq s b \quad & \quad |c|^+ b \leq s
\end{align*}
\]
Whence \( c \leq t \) for \( c \in \{ e^+, e^+, c \} \). Name restriction is no longer compatible stricto-senso; we have
\[
\begin{align*}
e^+ b \leq s(c(e^+ v 1))
\end{align*}
\]
The identity comes from the opening rule, where name restriction disappears. This is however sufficient to conclude that \( e^+ \leq t \).

The proof for parallel composition also has to be slightly adapted, to take care of the new side conditions in the rules, but also because of scope extrusion: in the second synchronisation rule, a name restriction appears, which was not there in CCS. Accordingly, one can prove
\[
\begin{align*}
c | b \leq s(c| (b v 1) v c^+ c)
\end{align*}
\]
Since we already obtained \( e^+ c \leq t \), this suffices to deduce \( c ^+ \leq t \).

Replication handled exactly as in the previous section (our proof of Lemma 8.4 is generic). The counterpart of Lemma 8.3 however requires more care, because of the condition on bound output transitions in the definition of \( s \). The problem is well-known: although bisimilarity is transitive in the \( \pi \)-calculus, the composition of two bisimulations is not always a bisimulation [29]. The reason appears explicitly in the following proof.

**Lemma 8.5** (Lemma 8.3 for \( \pi \)). In the \( \pi \)-calculus, the following functions are below \( t \)
\[
\begin{align*}
\sigma : R \mapsto \{ (P, Q) \mid \sigma \text{ injective name substitution, } P \models R Q \}
\end{align*}
\]
\[
\begin{align*}
j : R \mapsto \{ (P, R) \exists Q. P \models R Q, Q \models R \}
\end{align*}
\]
\[
\begin{align*}
k : R \mapsto \{ (P, Q) \exists P', Q'. P \models P', Q \models Q' \models Q \}
\end{align*}
\]
**Proof.** The function \( \sigma \) is actually compatible. In contrast, the squaring function \( j \) is not compatible in \( \pi \). We use second-order enhancements and symmetry arguments to reduce the problem to \( j b \leq sTj \). To prove this inclusion, let \( R \) be a relation and assume \( P \models b R Q \) and \( Q \models b R Q \) for some processes \( P, Q, R \). We have to show that \( \langle P, R \rangle \in sTj \). Further assume \( P \models b R' \).

- If \( \alpha \) is not a bound output, then \( b \alpha \leq \emptyset \) and the proof is standard: we first get \( Q' \) such that \( Q \models \alpha Q' \) and \( P' \models R' Q' \), and then \( R' \) such that \( R \models \alpha R' \) and \( Q' \models R' R' \). We finally check that \( P' \models \alpha R' R' \), and thus \( P' \models \alpha R' R' \).

- If instead \( \alpha = w / b \), then we know by assumption that \( b \# R \), but we would need \( b \# Q \) to perform the first step above. Since this might not be the case, we pick some name \( c \) fresh for both \( P, Q, R \), and use the permutation \( \sigma = (bc) \) to play a challenge on \( \pi \alpha c \) instead: we have \( P \models \pi \alpha c \alpha P' \) with \( c \# Q \), so that we get \( Q' \) such that \( Q \models c Q' \) and \( P' \models R' Q' \). Then since \( c \# R \), we get \( R' \) such that \( R \models c R' \) and \( Q' \models R' R' \). Since \( b, c \# R \), it follows that \( R \models \alpha R' \) and \( Q' \models R' R' \). Since \( \sigma \) is injective, we get \( \sigma P \models \alpha j R \sigma R' \). We have \( \sigma P \models R' \). Since \( \sigma \) is injective, we get \( \sigma P \models \sigma j R \sigma R' \). From \( \sigma P \) and \( c = 1 \) and \( \sigma \leq t \), we finally obtain \( P' \models \alpha R' R' \).

That \( k \leq t \) follows from \( j \leq t \) and the closure properties of \( t \) (Lemma 3.2).

**9. Respectful vs. compatible**

Before turning to parameterized coinduction, we discuss a historical peculiarity which has been causing some troubles since the introduction of up-to techniques, and which we can nicely resolve by using the companion function.

When Sangiorgi studied the bisimulation proof technique [28], he introduced the notion of respectful function to obtain compositionality results. With the present notation, a monotone function \( f : [X \rightarrow X] \) is respectful (for \( b \)) if for all \( x, y \in X \),
\[
\begin{align*}
x \leq y \text{ and } x \leq y \text{ entail } f x \leq f y .
\end{align*}
\]

Let \( b' \models b \land 1 \) (i.e., \( b' y = b y \land y \)). Without the assumption \( x \leq y \), respectability would be equivalent to compatibility (for \( b \)). With this assumption, it is equivalent to compatibility for \( b' \).

One can easily show that any compatible function (for \( b \)) is respectful, but some interesting respectful functions are not compatible. This is the reason why Sangiorgi needed this refinement. For instance the context closure function in CCS is respectful but not compatible. Hur et al. used respectability for the same reason [11].

In our own previous work [18, 23, 25], some of which with Sangiorgi, we found that the theory of plain compatibility was somewhat nicer to develop than that of respectability, so that we proposed to use the function \( b' \) rather than \( b \) when necessary (doing so is always possible). Although this was not their only reason, Parrow and Pohjola also chose a function \( b \) such that \( b = b' \) in their theory of up-to techniques for the psi-calculus [22].

In this paper, we used the most natural function \( b \) to define strong bisimilarity in CCS, and this function does not satisfy \( b = b' \). So how is it possible that we could obtain up-to context?

The point is that with the companion function, we do not need the up-to context function to be compatible stricto-senso. It just has to be below the companion function \( t \). For instance, in our proof for the parallel composition operation in CCS, we obtained \( e^+ b \leq bTc^+ \), which does not entail compatibility of \( e^+ \). This contrasts with the literature, where we would prove \( e^+ b' \leq b' c^+ \), i.e., that \( e^+ \) is respectful (compatible for \( b' \)).

That we can recover up-to context in CCS and \( \pi \) without switching to \( b' \) is not a coincidence: the greatest respectful function always coincides with the greatest compatible function:

**Proposition 9.1.** Let \( t' \) be the companion of \( b' = b \land 1 \). We have \( t' = t \) and \( b' = b^+ \)
\[
\begin{align*}
\text{Proof.} \quad \text{Any function compatible for } b \text{ is compatible for } b', \text{ whence } t \leq t' (\ast). \text{ Then we show } b \leq b' t (\ast\ast); \text{ we have } b' t = b t \land t; \text{ we get } b \leq b t \text{ using (5), and } b \leq t \text{ is just (4).}
\end{align*}
\]
To obtain \( t' \leq t \), we show that \( t' \) is compatible for \( b' \):
\[
\begin{align*}
t' b \leq t' b t \quad \text{(by (\ast\ast))}
\end{align*}
\]
\[
\begin{align*}
\leq b' t' \quad \quad \text{(compatibility of } t')
\end{align*}
\]
\[
\begin{align*}
\leq b' t \quad \quad \text{(by (\ast)) and idempotence of } t'
\end{align*}
\]
\[
\begin{align*}
\leq b \quad \quad \text{(by (\ast\ast))}
\end{align*}
\]

Finally, \( b t \leq b t t = b t \leq b t \text{ by (\ast\ast)}, \text{ idempotence of } t, \text{ and the fact that } b \leq b \). So that \( b t' = b' t = b t = b' \).

In other words, the historical tradeoff between \( b \) and \( b' \), or compatibility and respectability, is irrelevant. The functions \( b \) and \( b' \) lead to the same coinductive proof principle once enhanced with their companion. One can actually go even further and show that obtaining specific up-to techniques is equally hard with \( b' \) and \( b \); their (enhanced) second-order proof principles also collapse.

**Proposition 9.2.** Let \( B' \) be the second-order function associated to \( b' = b \land 1 \), and let \( T' \) be the companion of \( B' \) (so that \( \nu B' = t' = T' \)). We have \( T' = T \) and \( B' = B' \).
\[
\begin{align*}
\text{Proof.} \quad \text{The proof is slightly harder than the previous one; it is given in appendix. Note that unlike } b \text{ and } b', B \text{ and } B' \text{ cannot be compared in general.}
\end{align*}
\]
Independently from this work, Parrow and Weber gave a nice and intriguing characterisation of the largest respectful function [21], and thus, indirectly, of the companion. Translated in our setting their characterisation is as follows. Define the sequence \((b_\alpha)_\alpha\) of elements of \(X\) by transfinite induction on the ordinal \(\alpha\): 
\[ b_{\alpha+1} \triangleq b \circ b_{\alpha} \text{ and } b_\lambda \triangleq \bigwedge_{\alpha<\lambda} b_\alpha. \]
This sequence is decreasing and well-known to converge to \(v\). For all \(x \in X\), we have
\[ tx = \bigwedge_{x \leq b_\alpha} b_\alpha. \]

When \(x = \bot\), we recover the standard stratification of the greatest fixpoint; when \(x = v\), we recover Corollary 3.4.

10. Parameterized coinduction

Recall the coinductive proof principle, as provided by Knaster-Tarski’s theorem:
\[ x \leq y \leq by \]
\[ x \leq v \]

This approach has an important drawback: the need to define the coinductive predicate \((y)\) up-front. This does not match the standard practice, where the coinductive predicate is obtained incrementally from \(x\), by progressively extending it until it becomes a post-fixpoint. In the context of a paper proof, one can always gather mentally from standard practice, where the coinductive predicate is obtained incrementally from \(x\), by progressively extending it until it becomes a post-fixpoint. In the context of interactive formal proofs, one can always gather mentally from standard practice, where the coinductive predicate is obtained incrementally from \(x\), by progressively extending it until it becomes a post-fixpoint.

Parameterized coinduction is already provided by the companion. Following this idea, we give an alternative presentation of parameterized coinduction.

Let us first prove the following counterpart to (27):

**Theorem 10.2.** For all \(x, y \in X\), if \(y \leq b(y \lor x)\) then \(y \leq tx\).

**Proof.** Assume \(y \leq b(y \lor x)\) \((H)\), and let \(f : z \mapsto \bigvee_{\alpha \leq y} y\). This function maps the points above \(x\) to \(y\) and all other points to the bottom element. In particular, \(f(x) = y\), so that we have to show \(fx \leq tx\). We show a bit more, namely, that \(f \leq t\).

To this end, we use second-order coinduction up-to, and we prove \(f \leq B^2 f\), i.e., \(fb \leq btf\). Let \(z \in X\). If \(x \leq b\), then \(fbz = \bot \leq b(tfz)\). Otherwise, assume \(x \leq b\) \((H')\); we have
\[ fbz = y \quad \text{(by definition of } f \text{ and } (H') \text{)} \]
\[ \leq b(y \lor x) \quad \text{(by } H\text{)} \]
\[ = bt(fx \lor x) \quad \text{(by definition of } f\text{)} \]
\[ \leq bt(fb \lor b) \quad \text{(by } H'\text{)} \]
\[ = b(t(fb \lor b))z \]

We easily check that \(t(fb \lor b) \leq tf\) using Proposition 6.4, so that we have \(fb \leq btf\), as required.

Intuitively, \(tx\) contains everything that can safely be deduced from \(x\), not necessarily in a guarded way. In particular, \(x\) can be deduced from \(tx\). Instead, \(btx = G_b x\) is more restrictive and corresponds to guarded deductions only: we do not have \(x \leq ttx\) in general. With this intuition, the above theorem reads as follows: to deduce \(y\) from \(x\), one can assume \(y\) provided one switches to guarded deductions.

This leads us to the “proof system” given in Figure 4. The four rules are valid: if their premises hold, so do their conclusion. The first one is for initialisation: to prove that \(y\) is below the greatest fixpoint, deduce it from the empty context. The second rule is an axiom rule: if \(y\) belongs to the context \(x\), then we can deduce \(y\). The third one makes it possible to use any enhancement known to be below the companion. Typically, when some congruence closure is below the companion, this rule makes it possible to use coinduction (or inductive) reasoning.

Figure 4. A proof system for parameterized enhanced coinduction.

\[ \begin{array}{ll}
\text{y} \leq t \bot & \text{INIT} \\
\text{y} \leq v b & \text{DONE} \\
\text{y} \leq ft x & \text{UP TO f} \\
\text{y} \leq (y \lor x) & \text{COIND} \\
\end{array} \]

The fourth equation \((28)\), i.e., \([11, \text{Theorem 13}]\), becomes a simple corollary:

\[ tG_{b^2} = tb t. \]

This equation is proved in the appendix; it follows from Theorem 10.2 below. Note that with such a characterisation, Equation (28), i.e., \([11, \text{Theorem 13}]\), becomes a simple corollary:
A. Omitted proofs

Proof for Remark 3.8. First note that the function mapping a function to its companion is not monotone, so that the inequality \( t \leq t^* \) does not follow directly from \( b \leq b^1 \). Instead we show that \( t \) is compatible for \( b^1 \):

\[
tb^1 = tbt \leq bt = b^1 t
\]

Similarly for the converse inequality, we show that \( t^* \) is compatible for \( b^1 \):

\[
t^* b \leq t^* b^1 \leq b^1 t^* = bt^* = bt^* t^* \leq bt = b^1 t
\]

The second equality follows:

\[
b^1 t^* = b^1 t = bt = bt = b^1 t
\]

Proof of Proposition 9.2. First recall from Proposition 9.1 and its proof that \( t = t' \) and \( b \leq b^1 \). Let \( S \) be either \( T \) or \( T' \); we first show

\[
BS = B'S
\]

Let \( f, g : [X \to X] \), we have

\[
f \leq B'Sg \iff f' \leq b'Sg \quad \text{(by (13))}
\]

\[
\iff f' \leq bSg \quad \text{(because \( b \leq Sg \))}
\]

\[
\iff f \leq bSg \quad \text{(because \( b' \leq b \))}
\]

\[
\iff f \leq BSg \quad \text{(by (13))}
\]

Proof of Theorem 10.1. First note that we can strengthen the conclusion of Theorem 10.2 into \( y \leq bt \). Indeed, from the hypothesis \( y \leq bt(y \lor x) \) and the stated conclusion \( y \leq tx \) we deduce \( y \leq bt(tx \lor x) \); then we check that \( t(tx \lor x) = tx \).

Now recall that \( G_{b^1} x \) is the greatest fixpoint of the function \( z \mapsto bt(z \lor x) \). By monotonicity of \( bt \), \( bt \) is a post-fixpoint of this function: \( btx \leq bt(tx \lor x) \); thus \( btx \leq G_{b^1} x \). To prove \( G_{b^1} x \leq bt \), we use the aforementioned refinement of Theorem 10.2 and the fact that \( G_{b^1} x \) is a post-fixpoint, i.e., \( G_{b^1} x \leq b(G_{b^1} x \lor x) \).
References