An Equational Theory for Weak Bisimulation via Generalized Parameterized Coinduction

Chung-kil Hur
Seoul National University
Seoul, Republic of Korea

Paul He
University of Pennsylvania
Philadelphia, PA, USA

Yannick Zakowski
University of Pennsylvania
Philadelphia, PA, USA

Steve Zdancewic
University of Pennsylvania
Philadelphia, PA, USA

Abstract

Coinductive reasoning about infinitary structures such as streams is widely applicable. However, practical frameworks for developing coinductive proofs and finding reasoning principles that help structure such proofs remain a challenge, especially in the context of machine-checked formalization.

This paper gives a novel presentation of an equational theory for reasoning about structures up to weak bisimulation. The theory is both compositional, making it suitable for defining general-purpose lemmas, and also incremental, meaning that the bisimulation can be created interactively. To prove the theory’s soundness, this paper also introduces generalized parameterized coinduction, which addresses expressivity problems of earlier works and provides a practical framework for coinductive reasoning. The paper presents the resulting equational theory for streams, but the technique applies to other structures too.

All of the results in this paper have been proved in Coq, and the generalized parameterized coinduction framework is available as a Coq library.

1 Introduction

Coinduction is a powerful technique for reasoning about streams, computation trees, and other infinitary structures that are used widely in semantics and systems modeling. As such, coinductive proofs play a significant role in Coq developments like CompCert [Leroy 2009], FreeSpec [Letan et al. 2018], or Interaction Trees [Xia et al. 2020].

In such contexts, working with weak bisimulation (equivalence modulo hidden “internal” computation steps) is often desirable. However, naïve ways of applying coinduction, including its use for establishing weak bisimulations, suffer from lack of compositionality or incrementality. Compositionality allows the proof developer to create modular proofs using generic lemmas, while still ensuring sound coinductive reasoning. Incrementality lets them construct the bisimulation relation by accumulating parts of it during the proof, rather than having to posit the entire relation up front at the proof’s outset. Both of these properties are particularly useful in the context of mechanized formal proof.

The situation was improved by the introduction of the parameterized coinduction approach by Hur et al. [2013], and its implementation in the pacO library for Coq. The crux of the approach is to move away from specifying the greatest fixed point up front and instead to work with a predicate parameterized by “accumulated knowledge” that one can use during the construction of the proof to incrementally build the postfixed point. Hur et al. show that pacO supports reasoning up-to closures too, and they hinted that it might be pragmatic to systematically work with the greatest compatible closure (that is, the most general closure among a class satisfying good closure properties). This idea has been studied in greater length by Pous [2016], leading to the so-called companion approach, to which we compare ourselves in Section 7.

Despite these advances, there are still several difficulties with developing coinductive proofs in interactive theorem provers. First, the pacO reasoning principles are still too weak, resulting in cumbersome proofs. The limitation is particularly apparent when a proof nests two cofixed points: the inner cofixed point forgets all accumulated knowledge, leading to redundant reasoning. Second, the support for up-to reasoning remains either ad hoc or difficult to manipulate in existing approaches: we advocate here for internalizing and manipulating concretely defined closures, as opposed to the greatest compatible one. Finally, it still remains to package coinductive reasoning principles into “proof patterns” for weak bisimulation that are expressive and easy to work with in practice.

This paper addresses the above problems by making two technical contributions:

- We present an equational theory over streams that gives a novel axiomatic interface for working with weak bisimulations. This yields an “API,” realized by a set of lemmas, that helps users structure their coinductive proofs of weak bisimulation. This equational theory is a simplified (and self-contained) presentation of a formalization of the equational theory of interaction trees [Xia et al. 2020].
To prove the soundness of the equational theory, we introduce Generalized Parameterized Coinduction, gpaco, a backwards-compatible generalization of the paco framework. This new construction provides the ability to record previously available knowledge that has been accumulated during a coinductive proof, which solves paco’s issue with nested cofixed points. Additionally, it has intrinsic support for up-to reasoning, which, in contrast to the companion approach, allows for the creation of generic lemmas that aid in modular proof developments. We show that gpaco supports novel coinductive principles.

The rest of the paper explains these contributions in detail, working from gpaco to the equational theory. We first briefly review paco in Section 2 and highlight, by way of example, the shortcomings that motivate our generalized definition. Section 3 presents generalized parameterized coinduction, establishes its basic properties, and explains the reasoning principles that it justifies. We then incorporate “up-to closures” into the definition, again establishing the appropriate metatheory. Sections 4 and 5 apply gpaco to develop an equational theory for reasoning about (weak) bisimulations of streams with \( \tau \) (internal) events. Here we also present our novel proof rules for working with those bisimulations. Finally, Section 7 provides a comparison with related work, and, in particular, explains the deficiency of working with the companion.

The reasoning principles presented in this paper are applicable with little-to-no overhead in the Coq proof assistant through an extension of the paco library.\(^1\) All of the definitions, metatheory and examples presented here have been verified in Coq.\(^2\) However, none of it is specific to this proof assistant, and all results should be transferable to any other system providing support for coinduction.

2 Background: paco and a motivating example

2.1 Notations

In this and the following sections, we consider a complete lattice \((C, \sqsubseteq, \sqcup)\) and \(f \in C^{\text{max}} - C\), a monotone function over \(C\) that we refer to as a functor. The typical use case in our context will instantiate \(C\) with \(\mathcal{P}(T \times T)\) for some type \(T\) (i.e. the lattice of binary relations over \(T\)), but the theory applies to any such lattice. In our Coq formalization, the main lattice is the one of propositional relations over \(C : C \rightarrow C \rightarrow \text{Prop}\).

Write \(X_f\) for the set of postfixed points of \(f\), i.e. \(x \sqsubseteq f(x)\). Tarski’s theorem implies that \(X_f\) admits an upper bound. We write \(v.f\) for this upper bound. Additionally, this upper bound is the greatest fixed point of \(f\), i.e. in particular \(v.f = f(v.f)\).

\(^1\)The public link is redacted for the purpose of this submission. The material will be provided as an anonymous tarball for the reviewers.

\(^2\)Redacted as well

2.2 Parameterized Coinduction

We briefly recall the central idea behind parameterized coinduction and its reasoning principles. Intuitively, it consists in moving away from using \(v.f\) itself and instead conducting a proof toward some \(G_f \in C^{\text{max}} - C\) that is parameterized by some accumulated knowledge:

**Definition 2.1 (Parameterized greatest fixed point).** Define \(G \in (C^{\text{max}} - C)^{\text{max}} - (C^{\text{max}} - C)\) to be:

\[G_f \sqsubseteq v.(\lambda y. f(r \sqcup y))\]

Here, we think of \(r\) as the “knowledge” accumulated during a proof. The intuition and usefulness behind this definition is best illustrated by the equations it satisfies. The soundness of the approach comes from the fact that it coincides with the greatest fixed point when no knowledge has been accumulated.

**Lemma 2.2** (Init). \(v.f \equiv G_f \bot\)

The central coinduction principle, mapping to a strong variant of Tarski’s principle, is expressed as an unfolding lemma. It intuitively states that the coinduction hypothesis as well as the accumulated knowledge are accessible behind the guard, i.e. an iteration of the functor \(f\).

**Lemma 2.3** (Unfold). \(G_f \sqsubseteq f(r \sqcup G_f)\)

Finally, the accumulation principle is the key to allow for incremental coinductive proofs: one can enrich the currently accumulated knowledge at any point.

**Lemma 2.4** (Acc). \(y \sqsubseteq G_f \sqiff y \sqsubseteq G_f (r \sqcup y)\)

The technique has been a wild success, most notably in the context of the Coq proof assistant in which it has been implemented. It at once enabled both incremental and compositional reasoning principles, two improvements that are of particular value when conducting mechanized proofs. Notably, parameterized coinduction is also entirely compatible with automation, something that the native reasoning principles provided by Coq for coinduction prohibited in practice.

2.3 Example: paco’s shortcomings

The typical coinductive proof using paco reasoning aims to prove a goal of the form \(y \sqsubseteq v.f\). One starts by using Init to obtain \(y \sqsubseteq G_f \bot\), after which the proof proceeds by using Unfold and Acc interleaved with other steps of equational reasoning. Such incremental proofs are considerably simpler to construct in an interactive theorem prover. However, the paco lemmas falter in the presence of nested cofixed points: they lose too much information about the accumulated knowledge, leading to redundant and more awkward to construct proofs, a deficiency that becomes more problematic as the technique scales to reason about more complex systems.

To illustrate this phenomenon, consider the coinductive stream data type that might be used for instance to represent
the trace of a transition system. Such an object is a potentially infinite sequence of internal events, $\tau$, and external (or visible) events $\beta(n)$, terminated (if finite) by the $\epsilon$ marker. Here, for simplicity, we assume that visible events carry a natural number. We will sometimes omit the $\beta$ constructor and just write $n$ (especially in examples) to save space.

Here are some example streams:

$$
\begin{align*}
  s_0 &= 01\epsilon & \text{finite stream} \\
  s_1 &= \tau 0 r 1 \epsilon & \text{finite stream} \\
  s_2 &= 012 \ldots n(n + 1) \ldots & \text{infinite increasing stream} \\
  s_3 &= 0 r 1 \tau 2 \ldots n r(n + 1) \ldots & \text{infinite increasing stream} \\
  s_4 &= 0 1 0 1 0 1 \ldots & \text{infinite alternating stream} \\
  s_5 &= \tau \tau \tau \tau \tau \ldots & \text{silent divergence}
\end{align*}
$$

It is well-known that strong bisimulation is often too tight a relation to be relevant when studying such systems. One should instead work “up-to-tau,” which means that, when considering whether two streams are “the same,” we can disregard any finite number of $\tau$ steps on either side. This weak bisimulation matches terminal constructors and identical external events one-to-one, but also allows for a finite number of $\tau$ steps to be stripped away from either stream at any given point. We write $s \approx t$ to mean that $s$ is equivalent to $t$ up-to-$\tau$. For the examples shown above, we have $s_0 \approx s_1$ and $s_2 \approx s_3$, but no other distinct pairs of streams are weakly bisimilar.

We delay the full exposition of a formal definition of this relation to Section 4. Here, we simply observe that we can define $\approx$ as the greatest fixed point of a function, $\text{euttF}$:

$$
\text{euttF} : \mathcal{P}(\text{stream} \times \text{stream}) \rightarrow \mathcal{P}(\text{stream} \times \text{stream})
$$

We can think of $\text{euttF}$ as acting on a set of pairs of streams $Y$, which behaves as the “coinductive hypothesis” in this definition. $\text{euttF}$ is defined so that it satisfies several properties that characterize weak bisimulation. Among them, we have:

**Lemma 2.5** ($\text{euttF}$ Tau Left).

$$X \subseteq \text{euttF}(Y) \iff \{(rs, t) \mid (s, t) \in X\} \subseteq \text{euttF}(Y)$$

**Lemma 2.6** ($\text{euttF}$ Vis).

$$X \subseteq Y \iff \{(ns, nt) \mid (s, t) \in X\} \subseteq \text{euttF}(Y)$$

The first lemma states that, when reasoning backwards using goal-directed proof search, if we want to show that $\tau \cdot s$ is related to $t$ by $\text{euttF}(Y)$, it suffices to show that $s$ is related to $t$ by $\text{euttF}(Y)$—we can drop a $\tau$ from the left stream. The second lemma states that if two streams begin with the same visible event $n$, we can directly appeal to the coinductive hypothesis $Y$ to establish the relation.

With this setup, we can give an example proof using paco-style reasoning and see where it can be improved upon.

Consider the two infinite transition systems $s$ and $t$ depicted in Figure 1. They each visually encode the different states two streams can be in. A stream can change state through either an internal step or by emitting an event. We also consider additional equations we know over the states of the streams: the edge labeled by an equality sign represents definitional equality—we assume we have such an equation in our context. The bottom half of Figure 1 characterizes the same two streams, but as a system of equations.

Their behaviors can therefore be described as follows. Both streams consist of an infinite cycle alternating between the visible events 1 and 0. In the left stream, each iteration of these two events is separated by a silent step, while the right stream starts the new cycle immediately—embodied by the definitional equality between $t_0'$ and $t_1$. Finally, both streams have an initial state stepping into the cycle by emitting 0.

We wish to build a weak bisimulation between both corresponding upper states of $s$ and $t$, that is to prove that $s_0 \approx t_0$ and $s_1 \approx t_1$. The paco library is the perfect tool for such a task: we would like to build our proof incrementally as we
explore the underlying transition systems. Let us venture step by step into this task, depicted in Figure 2.

This minimal example highlights a deep problem in the existing reasoning principles: unused accumulated knowledge is always guarded again, i.e. sent back behind the guard. We see this in the proof at the point where we use \( \text{Acc} \) on \( \tau \). X we know that the pair of states are in \( (b) \) that greatly ease reasoning in cases such as the one depicted gpaco (and succinctly gen-

struction that still supports up-to reasoning, but crucially knowledge in the presence of nested accumulation lemmas: paco the desired pair, \((s_0', t_0')\), where we have \( X_0 \) directly available on the right hand side. The problem is that even though the knowledge \( X_0 \) is available at point \((b)\), we have to discard it to use Acc at point \((e)\), which forgets the fact that \( X_0 \) was available.

The impact of this loss of information shows up later, when trying to conclude for the pair of states \( (s_0', t_0') \). A natural solution, depicted at point \((d)\), is to simply blindly go through a new round of unfolding and stepping under the functor, using successively Lemma 2.5 and 2.6. Note that Lemma 2.5 alone is not enough to go under the functor, it does not act as a guard. However, by taking this step, we are repeating a part of the proof we already did: taking the transition that emits a 1 for both streams. This may seem innocuous on such a toy example, but may in general require reiterating an arbitrarily complex proof.

Intuitively however, we would like to simply ignore this \( \tau \) on \( s \) and conclude by using \( X_0 \), knowledge that we made available earlier in the proof. The first part of this intuition, the innocuousness of the \( \tau \) guard, is a particular case of a more general reasoning principle: reasoning up-to silent steps. We can indeed formalize this idea using paco, by proving the following lemma:

**Lemma 2.7** (\( G_{\text{euttF}} \) Tau Left).

\[ X \subseteq G_{\text{euttF}}(Y) \implies \{(r,s), (t,s) \in X \} \subseteq G_{\text{euttF}}(Y) \]

It precisely states that one can strip a \( \tau \) from the left hand side under a call to \( G_{\text{euttF}} \). Using this lemma at point \((e)\) in Figure 2, we can therefore reduce our goal to relating the desired pair, \((s_1, t_1)\). However this is useless in this case due to paco’s inability to remember previously available knowledge in the presence of nested accumulation lemmas: we know that the pair of states are in \( X_0 \), knowledge that was made available before, and yet we cannot access it to conclude.

To alleviate these difficulties, we introduce a new construction that still supports up-to reasoning, but crucially offers a finer grained management of available knowledge.

### 3 Generalized Parameterized Coinduction

In this paper, we introduce a new construct, dubbed the **generalized parameterized greatest fixed point** (and succinctly referred to as gpaco), that we show satisfies new principles that greatly ease reasoning in cases such as the one depicted in Figure 1. Our new construct builds on the so-called parameterized greatest fixed point introduced by Hur et al. [2013], and implemented in Coq through the paco library.

We extend the parameterized greatest fixed point in two ways. First, we refine its treatment of available knowledge by making a distinction between knowledge that is available, or “already unlocked” and knowledge that is guarded, or “must be unlocked” Maintaining this distinction dramatically simplifies incremental coinductive proofs. Second, we build in support for “up-to” reasoning, another powerful technique that lets us construct coinductive relations using closure operators.

#### 3.1 Generalized incremental reasoning

Recall our unsatisfactory proof of Figure 1. One core issue comes from the fact that while the accumulated knowledge is safely released after a guard, it does not internalize the fact that this knowledge became available. The first extension we introduce is to precisely take this observation into account: the parameterized greatest fixed point is now parameterized by two elements representing accumulated knowledge.

The generalized parameterized greatest fixed point \( \hat{G}_f \ r \ g \), also shortened to gpaco, therefore intuitively represents the greatest fixed point of the functor \( f \) with available accumulated knowledge \( r \) and guarded accumulated knowledge \( g \), which becomes available only after making progress by applying \( f \). We express this distinction in the following definition, which uses \( G_f \ r \).

**Definition 3.1** (Generalized parameterized greatest fixed point (first definition)). Define \( \hat{G}_f \ r \ g \defeq r \lor G_f (r \land g) \).

Note that if we pick \( r = \bot \), this definition degenerates to \( G_f \ r \ g \), which gives us the following soundness property. As before, we call it Init because it lets us begin a coinductive proof by moving into the gpaco realm.

**Lemma 3.2** (Init).

\[ \hat{G}_f \ \bot \equiv G_f \ \bot \equiv v, f \]

We can also return to vanilla parameterized coinduction from the generalized version:

**Lemma 3.3** (Final).

\[ r \lor G_f \ g \subseteq \hat{G}_f \ r \ g \]

These two lemmas mean in particular that gpaco is fully backwards compatible with paco: no changes in previous definitions or statements written with paco are required, and the new reasoning principles are available for properties defined in terms of \( \hat{G}_f \).

---

3We overload the lemma names like \( \text{Init} \) and \( \text{Acc} \) which are defined both for \( G_f \) and \( \hat{G}_f \). Which one is meant can easily be distinguished from the context.
The Base equation below embodies the fact that available knowledge is stored in gpaco. By definition, it is indeed trivial to see that \( r \) is immediately available for use:

**Lemma 3.4 (Base).**

\[
\begin{align*}
& r \subseteq \hat{G}_f \ r \ g \\
\text{Naturally, in order for Base to be sound, the incremental principle extends only the guarded knowledge:} \\
\text{Lemma 3.5 (Acc).} \\
& x \subseteq \hat{G}_f \ r \ (g \sqcup x) \iff x \subseteq \hat{G}_f \ r \ g
\end{align*}
\]

Finally, stepping under the guard makes the guarded knowledge available. Note that the pattern of accumulation ensures that we always have the invariant that \( r \subseteq g \), which is why erasing \( r \) here does not lose information. Lemmas 2.5 and 2.6 derive from this general reasoning principle, instantiated to the functor euttF.

**Lemma 3.6 (Step).**

\[
f(\hat{G}_f \ g \ g) \subseteq \hat{G}_f \ r \ g
\]

With the addition of the available knowledge parameter to gpaco and its new reasoning principles, we are closer to a more succinct proof of Figure 1 without the extraneous steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need a statement analogous to Lemma 2.7, in order to strip off a \( \tau \) steps required in the previous proof. However, we still need

\[
X_0 \subseteq \nu. \ euttF \ \\
\text{lem. 2.6} \\
X_1 \subseteq \hat{G}_{\text{euttf}} X_0 \ X_0
\]

\[
\text{Lemma 3.7 (G_{euttf} Tau Left, idealized).} \\
X \subseteq \hat{G}_{\text{euttf}} \ r \ g \implies \{(s, t) \mid (s, t) \in X\} \subseteq \hat{G}_{\text{euttf}} \ r \ g
\]

Note that this lemma does not hold with the definition of gpaco introduced in this subsection. We will get back to its proper statement, as well as its soundness, in Section 3.2, once we have extended gpaco with intrinsic support for up-to reasoning. Accepting temporarily this slight idealization, we showcase in Figure 3 a proof of the example from Section 2 eliminating the undesired repetition.

This proof illustrates how the extra parameter provides just the right degree of freedom to remember knowledge collected across nested calls to Acc. Here, the first use of Acc at point (a) doesn’t yet provide any more flexibility compared to the old proof. At point (b), however, the Step operation copies \( X_0 \) from the “guarded knowledge” parameter to the “available immediately” parameter. Later, at the second use of Acc at point (c), \( X_0 \) remains available, even as \( X_2 \) is placed under the guard. The payoff comes at point (d), where we can immediately use \( X_0 \).

### 3.2 Up-to reasoning: generalized paco with closure

The ability to construct coinductive proofs incrementally, as considered above, is one technique that is invaluable for working with coinduction in an automated theorem prover. Another crucial technique is the use of “up-to” reasoning principles, which enable more scalable and modular proofs. The basic idea is to define a closure operator \( \text{clo} \in C \rightarrow C \) that, given a relation \( X \), extends it to \( \text{clo}(X) \), a larger relation that accounts for regularity in the coinductive argument.

For example, the closure operator used for Lemma 3.7 is:

\[
t_{\text{L}}(R) = \{(r, s, t) \mid (s, t) \in R\}
\]

In this section, we develop the enhancements to gpaco necessary to reason using these closure operators.

Before we proceed, we briefly review the state-of-the-art up-to techniques. Pous [2016] characterizes valid closures as any function bounded by the greatest compatible closure, called the companion. Specifically, an up-to function \( \text{clo} \in C \rightarrow C \) is compatible with \( f \) if \( \text{clo} \circ f \subseteq f \circ \text{clo} \). The companion \( \text{cpn}_f \in C \rightarrow C \) is the join of all such compatible functions, which is again compatible with \( f \). Then, \( \text{cpn}_f \) admits more flexibility and up-to principles for coinduction: in particular, \( \text{clo}(\text{cpn}_f(r)) \subseteq \text{cpn}_f(r) \) for any (not necessarily compatible) function \( \text{clo} \subseteq \text{cpn}_f \). In practice, most useful up-to functions are bounded by the companion.

In our approach, instead of using the companion, we parameterize our construct with the upper bound of valid closures, which we call a base closure, in order to allow a more explicit construction of the fixed point. This generalization is essential in the development of our equational theory for weak bisimulation in Section 5.

**Definition 3.8 (Generalized parameterized greatest fixed point).** We redefine the previous \( \hat{G} \), adding a “base” closure \( \text{bclo} \in C \rightarrow C \) as the second argument:

\[
\hat{G}_f \ r \ g \overset{\text{def}}{=} \text{bclo}^*(r \sqcup G_{\text{bclo}} \ (r \sqcup g))
\]

where \( \text{bclo}^* \) is the reflexive transitive closure of \( \text{bclo} \).
\[s_0 \approx 0 \quad s'_0 \equiv r + s_1 \quad s_1 \equiv 1 \quad s'_1 \equiv 2 s'_0 \quad t_0 \equiv 0 \quad t'_0 \equiv r' + t_1 \quad t_1 \equiv 1 \quad t'_1 \equiv 2 t'_0\]

**Figure 4.** Two weakly bisimilar streams when \(r \approx r'\)

Note that by choosing the companion as a base closure, we get the equality \(G_f \cong g\). We introduce the following useful notation:

\[G_f^{bclo} \overset{\text{def}}{=} G_f \cap \mathbb{B}\]

Then we can use any up-to function \(clo\) bounded by \(bclo\), and in fact even larger ones bounded by \(G_f^{bclo}\).

**Lemma 3.10 (Closure).** If \(clo \subseteq G_f^{bclo}\), then \(clo(G_f^{bclo} r g) \subseteq G_f^{bclo} r g\)

Since \(bclo \subseteq G_f^{bclo}\), in the case \(clo = bclo\), it is always valid to use Closure, which will be marked as Closure*.

For the base closure, we require a condition that is weaker than compatibility.

**Definition 3.11 (Weakly compatible closure).** \(bclo \in \mathbb{C}\) is weakly compatible with respect to \(f\) if \(bclo \circ f \subseteq f \circ G_f^{bclo}\).

We can begin using generalized parameterized coinduction from usual parameterized coinduction:

**Lemma 3.12 (Intr).** If \(bclo\) is weakly compatible for \(f\), then \(G_f^{bclo} \perp \perp \subseteq G_f \perp \perp\)

We are now ready to amend Lemma 3.7: it holds, provided we instantiate the base closure parameter of \(G\) with \(t\), or any greater closure in the sense of Lemma 3.10. In particular, Section 4 will introduce one such base closure: transitivity up-to directed transitivity.

For a more involved example showing how reasoning up-to closures can help, consider the streams in Figure 4, which are a modified version of the example we saw earlier in Figure 1. Here, rather than \(s\) taking an extra \(r\) step, both streams go through intermediate transitions \(r\) and \(r'\) respectively. Moreover, rather than defining the streams using definition equality \(\approx\), we instead specify them via strong bisimulation \(\approx_s\). In the case that \(r\) and \(r'\) are known to be weakly bisimilar to each other, the resulting streams remain weakly bisimilar. However, in order to prove that this is the case, the weak bisimulation relation would have to contain all of the internal bisimilar states of \(r\) and \(r'\), and moreover, it would have to somehow incorporate the states related by the underlying strong bisimilarity relation too.

Similarly, when proving the equivalence up-to-tau of two streams, it is intuitively the case that if \(r \approx r'\) and we want to coinductively relate \(r + s \approx r' + t\), it suffices to relate \(s\) and \(t\)—we can ignore the weakly bisimilar prefixes and focus on proving the tails of the streams equivalent.

Up-to reasoning formalizes these intuitions. First, we define two closure operators:

\[\text{prefix}(R) = \{(h_1 + t_1, h_2 + t_2) \mid h_1 \approx h_2 \land (t_1, t_2) \in R\}\]

\[\text{bisim}(R) = \{(a, b) \mid \exists a', b', a \approx a', b \approx b' \land (a', b') \in R\}\]

Being able to prove \(s_0 \approx t_0\) and \(s_1 \approx t_1\) up-to bisim and prefix allows for a proof conducted parametrically in the assumption \(r \approx r'\), leading to a proof with complexity similar to the one for Figure 1.

Using the resulting set of reasoning principles provided by gaco, summarized in Figure 5, we can proceed with the proof of weak bisimilarity for Figure 4, that is \(s_0 \approx t_0\) and \(s_1 \approx t_1\). We use bisim as our base closure, a choice that will be grounded in Section 4.

By leveraging the reasoning principles up-to (strong) bisimilarity (bisim) and up to prefix (prefix), we can derive a proof extremely similar to the previous examples. The difference lies in the application of the Closure rules at five points in the proof. We first apply Closure* twice with bisim to rewrite \(s_0\), \(t_0\), \(s_1\), and \(t_1\). Next we apply Closure* again to replace \(s'_0\) and \(t'_0\) with \(r + s_1\) and \(r'_0 + t_1\) respectively. We then apply Closure with prefix to remove the weakly bisimilar prefixes \(r\) and \(r'\). Finally we apply Closure* with bisim again to rewrite \(s'_1\) and \(t'_1\). The remainder of the proof follows as before.

### 4 Up-to-tau bisimulation of streams

In the previous section we introduced gaco, a greatest fixed point predicate recording both the accumulated knowledge guarded by a constructor and its already accessible counterpart. We additionally extended the construction to internalize the support for up-to closure.

We have described the novel, richer reasoning principles derived from gaco. We now illustrate its practical use concretely by establishing a rich equational theory to reason about weak bisimilarity of interactive systems. We develop this case study using the datatype of potentially infinite streams of internal and external events, and study their equivalence up to internal steps.

The approach and the results being general, we present them in lattice theoretic notations, but all results are formalized in Coq.

#### 4.1 Streams

The codata considered is the same type of potentially finite streams of internal and external events introduced earlier in the paper. Formally, we define stream where:

\[\text{stream} \overset{\text{def}}{=} \{\epsilon\} \cup \{r \cdot s \mid s \in \text{stream}\} \cup \{\beta(n) \cdot s \mid s \in \text{stream}, n \in \mathbb{N}\}\]
bclo weakly compatible for $f$
\[ \frac{G^{bclo}_f \bot \bot \subseteq G_f \bot}{x \subseteq \hat{G}^{bclo}_f \ r \ (g \sqcup x)} \frac{\bot \subseteq \hat{G}^{bclo}_f \ r \ g}{x \subseteq \hat{G}^{bclo}_f \ r \ g} \] 

INIT $r \in \hat{G}^{bclo}_f \ r \ g$
BASE $r \sqcup G_f \ g \subseteq \hat{G}^{bclo}_f \ r \ g$
FINAL $f(\hat{G}^{bclo}_f \ r \ g) \subseteq \hat{G}^{bclo}_f \ r \ g$
STEP
\[ \frac{\bot \subseteq \hat{G}^{bclo}_f \ r \ g}{\bot \subseteq \hat{G}^{bclo}_f \ r \ g} \]

\[ \text{base} = \mathsf{true} \wedge (s, t) \in \mathsf{bisim} \ f \ L \ r \ Clo X \} \]
\[ \text{final} = \mathsf{false} \wedge (s, t) \in \mathsf{bisim} \ f \ L \ r \ Clo X \} \]

\[ \text{step} = \mathsf{false} \wedge (s, t) \in \mathsf{bisim} \ f \ L \ r \ Clo X \} \]

\[ \text{closure} = \mathsf{false} \wedge (s, t) \in \mathsf{bisim} \ f \ L \ r \ Clo X \} \]

\[ \text{closure}^* = \mathsf{false} \wedge (s, t) \in \mathsf{bisim} \ f \ L \ r \ Clo X \} \]

**Figure 5.** Proof rules for generalized parameterized coinduction

An element of the resulting type stream is hence a potentially infinite trace consisting of internal steps, represented as $\tau$ constructors, and visible events, emitting natural numbers, represented as $\beta$ constructors. Such a datatype can for instance be thought of as the observable trace of an interactive program's execution.

We fix the lattice of interest to $\mathcal{P}(\text{stream} \times \text{stream})$ in the rest of the paper.

Defining a concatenation operation over streams, concat, is straightforward: let concat $\triangleq \nu$.concatF where

\[
\text{concatF \ concat} \triangleq \lambda s. k. \ \begin{cases} \\
| \quad \epsilon \Rightarrow k \\
| \quad \tau \cdot s \Rightarrow \tau \cdot (\text{concat} \_ s \ k) \\
| \quad \beta(n) \cdot s \Rightarrow \beta(n) \cdot (\text{concat} \_ s \ k) \\
\end{cases}
\]

We write $s \# t$ for concat $s \ t$.

Reasoning about these streams naturally requires to prove that concat respects an equivalence relation over streams, which justifies reasoning principles such as: $s \approx t \implies s \# k \approx t \# k$. The usual notion of Leibniz equality is inadequate when manipulating codata-types. Instead, the standard equivalences used to reason about such streams are the notions of strong and weak bisimulations.

### 4.2 Bisimulation, equivalence up-to $\tau$

A natural equivalence relation over stream is to require the shape of both streams to match exactly, systematically pairing the head constructors. This coinductive relation, known as strong bisimulation, is convenient to work with, but too restrictive in practice. Indeed, it not only observes the visible events two systems emit when comparing them, but also ensures that their internal steps match as well: in a sense, it is a timing-sensitive equivalence of processes.

Equivalence up-to-$\tau$ is a form of weak bisimulation, a coarser relation than strong bisimulation. It ignores any finite amount of internal steps a process may take before reaching its next external event. This relation is much more useful in practice, and is notably the de facto standard used in verified compilation to express the semantic preservation criterion [Leroy 2009; Tan et al. 2016].

Equivalence up-to-$\tau$ has to be careful not to relate the infinite sequence of $\tau$ with all streams. This is achieved by an inductive-coinductive definition: the functor bisimF whose greatest fixed point we take is itself defined recursively, but as a smallest fixed point. This nested structure makes it particularly delicate to work with without a carefully crafted metatheory. Moreover, because strong and weak bisimilarity have some common structure, it is beneficial for proof engineering purposes to share as much of their common metatheory as possible.

We demonstrate in this section how introducing a parameterized version of the weak bisimulation relation allows us to derive a rich equational theory that alleviates the pain of working with nested inductive-coinductive definitions. Our new construction, gpaco, is instrumental to the proofs in this theory.

### 4.3 A family of bisimulations

While weak bisimulation is the core relation we care about, several related relations are relevant to prove our equational theory. As a way to factor work, we start by defining in Figure 6 $\text{bisim}$, a family of relations over streams. Let us for now ignore its three parameters and focus at a high level on the function $\text{bisim} \_ \ X$. We use the fix keyword as a notation to express $\text{bisim}F$ itself is defined as a smallest fixed point.

There are five ways we may relate two streams: 1. by matching $\epsilon$ constructs, 2. by matching $\tau$ and co-recurring, 3. by matching identical $\beta$ and co-recurring, 4. by stripping a $\tau$ from the left and recursing or 5. by stripping a $\tau$ from the right and recursing. Note the use of a recursive call when stripping $\tau$ in the asymmetric cases (4) and (5): if we were to iterate co-recursively, then an infinite co-recurcive chain of application of rule (4) would relate the silently diverging stream to any stream.
The three parameters to $\text{bisim}_F$ refine the way these rules can be used to derive different relations. The boolean $b_L, b_R$ flags enable or disable rules (4) and (5) respectively. The $\text{clo}_\beta$ parameter, of type $\mathcal{P}(\text{stream} \times \text{stream}) \rightarrow \mathcal{P}(\text{stream} \times \text{stream})$ is slightly more subtle. When matching two external events by rule (3), one does not have to relate the remaining of the streams with respect to just a co-recursive call, but instead can first quotient them by $\text{clo}_\beta$.

The practical use of the closure parameter will be delayed to Section 5 where it will be instrumental in deriving the necessary reasoning principles. For now, we set the $\text{clo}_\beta$ parameter to the identity closure $\text{id}$ in order to define the high level relations we are interested in. It is straightforward to check that $\text{bisim}_F b_L b_R \text{clo}_\beta$ is monotone for any monotone $\text{clo}_\beta$, in particular for $\text{id}$. We therefore can define the greatest fixed point $\text{bisim}_F b_L b_R$ using $\text{paco}$.

We are now ready to derive concrete relations. First, if both asymmetric rules are disabled, we have to exactly match all constructors: this corresponds to strong bisimulation.

**Definition 4.1** (Strong bisimulation).

$$s \equiv t \overset{\text{def}}{=} \text{bisim false false } s t$$

At the opposite side, equivalence up-to-tau is defined by allowing both rules: it is always fine to strip away finite amounts of $r$’s on either side:

**Definition 4.2** (Equivalence up-to-tau).

$$s \approx t \overset{\text{def}}{=} \text{bisim true true } s t$$

Finally, a third relation is often useful. By allowing only one of the rules, we get an asymmetric relation expressing that a stream is up-to-tau bisimilar to another, but contains more $\tau$:

**Definition 4.3** (Over-approximation up-to-tau).

$$s \triangleright t \overset{\text{def}}{=} \text{bisim false false } s t$$

Notice the following subrelation inclusions: $\equiv \subseteq \triangleright \subseteq \approx$.

Unfortunately, the inductive-coinductive nature of weak bisimulation in particular makes a property as elementary as transitivity already a challenge to prove. The standard approach is to seek stronger reasoning principle by introducing up-to techniques. We first consider reasoning up to transitive closure.

### 4.3.1 Transitive closure of the bisimilarity relations

The native reasoning principle on streams only allows us to step through the functor $\text{bisim}_F$, forcing us systematically to nest an induction to account for possible bounded stripping of $rs$, which often requires a clever generalization of the statement for it to hold inductively. Reasoning up-to transitive closure enables a new reasoning principle: when attempting to prove that two streams $(s_1, s_2)$ belong to a relation $r$, it may be sound in appropriate contexts to simply substitute $s_1$ or $s_2$ for other bisimilar streams.

This intuition is formalized by introducing a family of transitive closures parameterized by four booleans flags:

**Definition 4.4** (Transitive closure up to bisimilarity).

$$(s_1, s_1') \in \text{bisim}_F b_L b_R \quad (s_2, s_2') \in r \quad (s_2, s_2') \in \text{bisim}_F b_L' b_R'$$

$$s_1, s_2) \in \text{bisim}_F \text{trans}_\text{clo} b_L b_R b_L' b_R'$$

Each pair of flags defines the instances of $\text{bisim}_F$ that are allowed to be used to substitute for the left and right streams. These closures are not all safe to use in arbitrary contexts. Indeed, by setting all flags to true, we allow arbitrary rewriting up-to-tau:

**Definition 4.5** (Undirected transitive closure).

$$U \overset{\text{def}}{=} \text{bisim}_F \text{trans}_\text{clo} \text{true true true true}$$

Let us emphasize why such arbitrary, undirected, up-to-tau rewriting provided by $U$ is an unsound principle in general. Recall that a coinductive proof is in essence constructing a cycle by being only allowed to invoke the coinduction hypothesis once below a guard. In our case, $U$ could hence be misused to introduce a $\tau$ constructor that would then be used as guard, allowing for circular reasoning. To illustrate the problem concretely, let us assume for a moment that the precondition of the $\text{Closure}$ principle from Figure 5 is available for $U$. The following proof would then be valid:

$$\begin{align*}
0e \approx 1e & \overset{\text{bin}}{\iff} (0e, 1e) \in \hat{G}_{\text{outff}}(\emptyset) \\
\text{Acc} & \iff (0e, 1e) \in \hat{G}_{\text{outff}}(\emptyset) \{(0e, 1e)\} \\
\text{Closure}(U) & \iff (r0e, r1e) \in \hat{G}_{\text{outff}}(\emptyset) \{(0e, 1e)\} \\
\text{Step} & \iff (0e, 1e) \in \hat{G}_{\text{outff}}(\{(0e, 1e)\}) \\
\text{Base} & \iff (0e, 1e) \in \{(0e, 1e)\} \quad \square
\end{align*}$$

This minimal example show-cases how this unrestricted up-to closure principle could introduce $\tau$ constructors that would then be used as guards to wrongly justify the use of the coinductive hypothesis. Thankfully, applying $\text{Closure}(U)$ is prohibited. Note however that had we justified the use of the coinductive hypothesis by a $\beta$ guard, the rewriting would have been harmless.

We will come back to $U$ in more detail by considering a context-sensitive up-to technique in Section 5. But let us focus for now on a better behaved instance:

**Definition 4.6** (Directed transitive closure).

$$D \overset{\text{def}}{=} \text{bisim}_F \text{trans}_\text{clo} \text{true false true false}$$

The $D$ closure disables the second flag used in the setting of each bisimulation considered. This means that a stream may be substituted by a bisimilar one, only if the new one contains fewer $rs$ than the previous one. It is intuitively clear that this substitution is always sound since it cannot introduce a guard. Moreover, this transitivity principle is in practice the most general one that we shall consider. It will
be the instance of the base closure that we will provide to gpaco in the construction we introduce in Section 5.

This soundness and generality are expressed by proving that \( D \) provides a sound up-to reasoning principle with respect to \( \approx \). This soundness holds in the sense that \( D \) satisfies the precondition from Lemma 3.12 with respect to the functor euttF \( \defeq \text{bisim} \). True.

Lemma 3.12 allows us to move from a proof of a paco predicate, \( \approx \) being the one of concern, to a gpaco counterpart setup with \( D \) as the base closure.

**Lemma 4.7** (Initialization for \( D \) with respect to euttF). For any \( \text{clo}_\beta \) monotone such that \( D \circ \text{clo}_\beta \subseteq \text{clo}_\beta \circ D \), \( D \) is weakly compatible for euttF \( \text{clo}_\beta \).

We can at this stage already establish a certain number of facts about our instances of bisim. By picking in particular \( \text{clo}_\beta \equiv \text{id} \), the closure used in the definition of euttF, we can derive the following reasoning principle by applying Closure*.

**Theorem 4.8** (\( \equiv \) is a congruence for \( \geq \)).

\[
\frac{s' \geq s}{s' \equiv s} \quad \frac{s' \equiv t'}{t' \geq t} \quad \frac{s \equiv t}{s' \equiv t'}
\]

We then prove that bisim defines equivalence relations:

**Lemma 4.9.** \( \equiv \) and \( \approx \) are equivalence relations. \( \geq \) is reflexive and transitive.

And finally show that \( \text{bisim} \ h_L \ h_R \) is a congruence for each constructor of euttF.

4.3.2 Concat closure

Proving the monoidal laws and congruence rules relating concat to weak bisimulation is greatly simplified by a second reasoning principle: the ability to reason up-to prefix. When attempting to relate two streams defined as concatenations, it should be possible to discharge their prefixes by proving they are bisimilar. The following closure captures this reasoning principle:

**Definition 4.10** (Concat closure).

\[
\frac{h_1 \equiv h_2 \quad (t_1, t_2) \in r}{(h_1 + t_1, h_2 + t_2) \in C \ r}
\]

The soundness of the closure is embodied by showing that Lemma 3.10 can be instantiated for \( C \) with respect to euttF, with \( D \) for the base closure:

**Lemma 4.11** (Compatibility of \( C \) with respect to euttF). For any \( \text{clo}_\beta \) monotone such that \( C \circ \text{clo}_\beta \subseteq \text{clo}_\beta \circ C \) and id \( \subseteq \text{clo}_\beta \), we have \( C \subseteq C_{\text{euttF}} \circ \text{clo}_\beta \).

Lemma 4.11 essentially states that all instances of bisim are congruences for concat in the first argument. In particular we can prove that \( \equiv \) is a congruence for concat:

**Theorem 4.12** (\( \equiv \) is a congruence for concat).

\[
\frac{h_1 \equiv h_2 \quad t_1 \equiv t_2}{h_1 + t_1 \equiv h_2 + t_2}
\]

With these tools in hand, we can prove the expected monoidal laws. In particular, Theorem 4.12 greatly simplifies the proof of associativity.

**Theorem 4.13** ((\( \text{stream}, + \)) forms a monoid).

\[
s + \epsilon \equiv s \quad s + \epsilon \equiv s \quad (r + s) + t \equiv r + (s + t)
\]

5 An equational theory for weak bisimulations

Section 4 introduced the stream datatype and two equivalence relations upon it: a strong bisimulation that constrains them to be structurally identical, and a weak bisimulation that quotient them up-to finite amount of internal steps. We have shown that two reasoning principles may be proved sound when reasoning about weak bisimulations: up-to transitivity with respect to addition of taus, \( D \), and up-to concat closures, \( C \).

However, even with the support from gpaco, reasoning about streams remains a technical challenge. In particular, we noticed that up-to transitivity with respect to general equivalence up-to-tau, \( \mathcal{U} \), is sound in contexts guarded by a \( \beta \), but not when guarded by a \( \tau \).

In order to alleviate these difficulties, we abstract away from the low-level use of gpaco and define through this section a new context-sensitive weak bisimulation relation, euttG. We prove that this relation satisfies a rich equational theory, notably supporting context-sensitive up-to techniques, and is sound with respect to weak bisimulation. By doing so, we hence internalize much of the complexity inherent to coinductive reasoning over weak bisimulation and provide an interface exposing the higher level reasoning principles specific to weak bisimulations of streams.

5.1 A context-sensitive weak bisimulation

We leverage the expressivity of gpaco to define the parameterized weak bisimulation euttG \( r_\beta \ r \ r_\beta \ r_\beta \). Before getting to its formal definition, we sketch the intuition it carries. The relation takes four parameters, each of type \( \mathcal{P}(\text{stream} \times \text{stream}) \), which correspond respectively to information that has been unlocked by a visible step or an internal step, or that remains guarded behind a visible step or an internal step.

The key idea in distinguishing the kind of constructor that has released or still locks the information is to allow for context-sensitive up-to techniques. Indeed, an incremental coinductive proof can be thought as a game of exploration whose goal is to close all paths explored by coming back to a previously explored state. By substituting a stream for a weakly bisimilar one, we may compromise all states reached by taking \( \tau \) steps, but we remain certain that a cycle is found
if we get back to a state reached under a $\beta$ step. As such, $\beta$
guards are stronger than $\tau$ guards when reasoning up-to-tau.

The main tool we will use to enable more reasoning principles under $\beta$
guards than $\tau$ guards is the $\text{clo}_\beta$ argument introduced in the definition of $\text{bisim}$, Figure 7, and which has been left unexploited through Section 4. Recall that this parameter is a closure up-to which the remaining streams may be quotiented during the corecursive call under a $\beta$
constructor. The closure we consider is defined as follows:

**Definition 5.1** (Closure for external events).

$$\mathcal{V}_{g_\beta} r \overset{\text{def}}{=} \mathcal{C}_{\text{euttf}}^{\text{D}} \mathcal{V}_{g_\beta} \Pi (U(r \cup g_\beta)).$$

The closure $\mathcal{V}_{g_\beta}$ is best understood right to left. At its
core, it simply extends the relation $r$ with the $\beta$ guarded
knowledge $g_\beta$. Since it will only be accessible under $\beta$
guards, it is also sound to close this knowledge up to undirected
transitivity, $U$, to allow for arbitrary rewriting by weak
bisimilarity. Finally, by definition of $\text{bisim}^\mathcal{F}$, using $\mathcal{V}_{g_\beta}$ in
place of the $\text{clo}_\beta$ argument permits its use right as we strip
off a pair of $\beta$ constructors. Specifically, if the goal is of the
form $\beta(n) \cdot s \approx \beta(n) \cdot t$, then $\mathcal{V}_{g_\beta}$ can be used to relate $s$ and $t$.
However, we sometimes want to delay the use of this closure:
say the goal is of the form $\beta(n) \cdot p + s \approx \beta(n) \cdot p + t$, we need
to first reason up-to concatenation and only then use $\mathcal{V}_{g_\beta}$ to
relate $s$ and $t$. Wrapping the whole closure into a call to
$\text{gpaco}$ is a convenient way to make this possible.

We now turn to the definition of $\text{eutgT}$ itself:

**Definition 5.2** (Parameterized weak bisimulation).

$$\text{eutgT} (r, \{g_\beta\} g_t) \overset{\text{def}}{=} \mathcal{V}_{g_\beta} (U(r) \cup g_t).$$

The definition of $\text{eutgT}$ is a slightly intimidating instance of
$\text{gpaco}$. Let us walk through each of its arguments. First,
the base closure provided is $D$: in any context, it is sound
to work up to directed transitivity. Now since both $r_\beta$ and
$r_\tau$ are information that has been unlocked previously, their
union is provided as accessible, except that, as in the case of
$g_\beta$ under $\mathcal{V}$, the $\beta$ unlocked knowledge is additionally
closed by $U$ — undirected transitivity. The functor whose
greatest fixed point we take is naturally $\text{euttf}$; going under
the functor hence guarantees that we go either under a $\tau$ or
a $\beta$ guard. We therefore set $g_\beta$ to be always unlocked under
the functor, as expressed by its position as last parameter of
$\text{gpaco}$. Finally, the additional knowledge $g_\beta$ is ensured to be
only unlocked when the functor is applied by going under $\beta$
guards by being provided as a parameter to $\mathcal{V}$ in the closure
passed to $\text{euttf}$.

Having motivated the definition of $\text{eutgT}$ by the intuitive
reasoning principles it should satisfy, we formalize these
principles in the following subsection.

### 5.2 An equational theory for $\text{eutgT}$

The interface provided by our theory is summarized by the
set of rules described in Figure 7. They are split into four
categories. The soundness rules relate equivalence up-to-tau and
$\text{eutgT}$. The knowledge manipulation rules provide the
core coinductive principles specialized to weak bisimulation.
The stream processing rules give specialized principles to
step under $\text{euttf}$ constructors. Finally, we provide support
for three up-to reasoning principles. All rules maintain the
following implicit invariant for $\text{eutgT}$: $r_\beta \subseteq r_\tau \subseteq g_t \subseteq g_\beta$.

**Soundness** The relation between $\text{eutgT}$ and $\approx$ is similar
to the one between $\text{paco}$ and $\text{gpaco}$: it is an intermediary
construct one transits to in order to conduct a proof.

The soundness of the overall approach is hence encapsulated
into two rules. First, the Init rule states that one can always
move during a proof of weak bisimulation into the
$\text{eutgT}$ realm by assuming no initial knowledge.

**Theorem 5.3** (Init).

$$(s, t) \in \text{eutgT} 0 0 0 0 \implies s \approx t$$

Using Init, we can hence start a $\text{eutgT}$-based proof. Conversely,
since $\text{eutgT}$ is purely an intermediary to conduct
proofs about weak bisimulation, Final is key to invoke any
pre-established $\approx$-equation: for any state of accumulated
knowledge, $\text{eutgT}$ always contains $\approx$.

**Theorem 5.4** (Final).

$$s \approx t \implies (s, t) \in \text{eutgT} r_\beta r_\tau g_\beta g_t$$

**Knowledge manipulation** The $\text{eutgT}$ relation shields the
user from its internals as much as possible by providing its
own reasoning principles with respect to the four knowledge
arguments it carries. First, the Base case echoes its $\text{gpaco}$
counterpart by giving access to all unlocked knowledge.

**Theorem 5.5** (Base).

$$(s, t) \in r_\beta \cup r_\tau \implies (s, t) \in \text{eutgT} r_\beta r_\tau g_\beta g_t$$

The accumulation theorem is once again key to make
parameterized coinductive reasoning possible. It states that
in order to prove that a set $x$ of pairs of streams belongs to
$\text{eutgT}$, one can extend the guarded knowledge by assuming
that $x$ is contained in this knowledge:

**Theorem 5.6** (Acc).

$$x \subseteq \text{eutgT} r_\beta r_\tau g_\beta g_t \iff x \subseteq \text{eutgT} (r_\beta \cup x) (g_\beta \cup x)$$

**Stream processing** Three principles allow us to process
each of the stream constructors. Naturally, it is trivial to
show that terminating streams can be matched.

**Theorem 5.7** (Ret).

$$(\epsilon, \epsilon) \in \text{eutgT} r_\beta r_\tau g_\beta g_t$$

Internal events can be consumed on each side, which grant
access to the $\tau$ guarded knowledge.

**Theorem 5.8** ($\tau$ step).

$$(t, s) \in \text{eutgT} r_\beta g_t g_\beta g_t \implies (s \cdot t, t \cdot t) \in \text{eutgT} r_\beta r_\tau g_\beta g_t$$
Finally, visible steps propagate the guarded knowledge to all parameters.

**Theorem 5.9 (β step).**

\[
(\mathcal{S}, t) \in \text{euttG} \quad \text{β} g_β g_β g_β \quad \implies \quad \beta(n) \cdot s, \beta(n) \cdot t \in \text{euttG} r_β r_τ g_β g_τ
\]

**Up-to reasoning** Finally, three up-to reasoning principles are supported. As developed in Section 4, directed transitive closure and concatenation closure are sound in all contexts. This gets reflected in the simplicity of rules \text{TRANS}D and \text{CONCAT}: one can simply make a call to the corresponding closure at any time.

**Theorem 5.10 (Directed transitive closure).**

\[
(\mathcal{S}, t) \in \mathcal{D}(\text{euttG} r_β r_τ g_β g_τ) \quad \implies \quad (\mathcal{S}, t) \in \text{euttG} r_β r_τ g_β g_τ
\]

**Theorem 5.11 (Concat closure).**

\[
(\mathcal{S}, t) \in \mathcal{C}(\text{euttG} r_β r_τ g_β g_τ) \quad \implies \quad (\mathcal{S}, t) \in \text{euttG} r_β r_τ g_β g_τ
\]

The third principle, undirected transitive closure, is more interesting. We internalize the intuition that it is only sound while guarded by \(\beta\) guards by overwriting all weakly available and guarded knowledge by the strongly available one:

**Theorem 5.12 (Undirected transitive closure).**

\[
(\mathcal{S}, t) \in \mathcal{U}(\text{euttG} r_β r_τ g_β g_τ) \quad \implies \quad (\mathcal{S}, t) \in \text{euttG} r_β r_τ g_β g_τ
\]

We now illustrate a use of this interface.

### 5.3 Practical use of euttG

Consider the following two streams:

\[
\begin{align*}
\mathcal{S}_0 &\approx 0 s_0 \quad s_0' \approx r + s_1 \quad s_1 \approx 1 s_1' \quad s_1' \approx 2 s_0' \\
\mathcal{T}_0 &\approx 0 t_0 \quad t_0' \approx r' + t_1 \quad t_1 \approx 1 t_1' \quad t_1' \approx 2 t_0'
\end{align*}
\]

This example differs from Figure 4 in that each of the states are related to one another by weak bisimilarity. To prove that \(\mathcal{S}_0 \approx 0\) and \(\mathcal{T}_0 \approx 0\), the same proof as before using just \(g\) and \(\mathcal{S}_0'\) and \(\mathcal{T}_0'\) for \(\mathcal{S}_0\) and \(\mathcal{T}_0\), respectively.

\[
\begin{align*}
\mathcal{S}_0' \approx 0 s_0' \quad s_0' ' \approx r + s_1' \quad s_1' \approx 1 s_1' ' \quad s_1' ' \approx 2 s_0' '\n\mathcal{T}_0' \approx 0 t_0' \quad t_0' ' \approx r' + t_1' \quad t_1' \approx 1 t_1' ' \quad t_1' ' \approx 2 t_0' '
\end{align*}
\]

will not work, since we need to use \(\mathcal{U}\), a context-sensitive closure. However, the proof remains straightforward using \(\text{euttG}\), as depicted in Figure 8.

Notice in particular how \(\text{TRANS}\) allows us to rewrite up-to-tau equations, at the cost each time of losing the knowledge locked behind a \(r\) guard.

### 5.4 Essential need for the base closure

We show that the companion closure is inconsistent with the rules of \(\text{euttG}\), so that it cannot be used as a base closure. To this end, for any definition of \(\text{euttG}\) satisfying the rules in Figure 7, suppose that it is closed under the companion \(\text{cpn}_F\) for \(F = \text{bisimF} b_L b_R \text{clo}_\beta\) with arbitrary \(b_L, b_R, \text{clo}_\beta\):

\[
\begin{align*}
\text{cpn}_F(\text{euttG} r_β r_τ g_β g_τ) \subseteq \text{euttG} r_β r_τ g_β g_τ
\end{align*}
\]
We derive a contradiction. Let \( X = \{(1\epsilon, 2\epsilon)\} \) and \( Y = \{(01\epsilon, 02\epsilon)\} \). First, for \( \mathbb{T} = \text{stream} \times \text{stream} \), we have:

\[
\text{cpn}_f(Y) = F(\mathbb{T}) \quad (2)
\]

The proof of (2) is given in Appendix A. Then we have:

\[
\begin{align*}
X \subseteq U(F(X)) & \quad \text{(since } (1\epsilon, 2\epsilon) \in F(X) \text{)} \\
\subseteq U(F(\mathbb{T})) = U(\text{cpn}_f(Y)) & \quad \text{(by (2))} \\
\subseteq U(\text{cpn}_f(\text{euttG } 0 0 X 0)) & \quad \text{(by } \beta \text{-Step)}
\end{align*}
\]

Therefore, by Acc, we have \( X \subseteq \text{euttG } 0 0 0 0 \) and thus, by \( \text{inut}, 1\epsilon \approx 2\epsilon \), which is a contradiction.

The root of the issue is that the companion construction contains non-structural junks when provided a false assumption like \( Y \) above. Where we would want \( \text{cpn}_f(Y) \) to contain exactly the pairs of streams equivalent modulo \( Y \), it also ends up containing nonsensical pairs such as \((1\epsilon, 2\epsilon)\).

6 Implementation in the Coq proof assistant and scaled case-study

We implemented gpaco and its theory as described through Section 3 in the Coq proof assistant. The formalization is built as an extension of the paco library and available at: \text{REDACTED FOR ANONYMITY}.

Since the implementation builds directly on top of paco, it is fully backward compatible: the new gpaco reasoning principles are applicable to any coinductive object defined via paco, with no change in the definitions. As was the case with the original library, we provide high level tactics mapping to each reasoning principle described in Figure 5.

6.1 Large Scale Case-Study: Interaction Trees

For sake of exposition and self-containment, we have presented here a case-study built on streams and their monoidal structure. The motivation for the development of this technique however stemmed from a more complex application: interaction trees [Xia et al. 2020] are a coinductive structure similar to streams, but branching in the sense that the visible events are followed by a continuation over the type of the emitted event. Interaction trees can be equipped with a bind operation similar to the concat operation, and proved to form a monad.

We have applied the techniques described in this paper to derive an axiomatic interface to reason up-to-tau about interaction trees. This layer of abstraction has then been heavily used to reason about this structure, and proved instrumental in alleviating the induced difficulty.

The corresponding formal development can be browsed at \text{REDACTED FOR ANONYMITY}.

7 Discussion and Related work

\textbf{Paco and Companion} We start by discussing how our contribution builds on existing works, namely parameterized coinduction (Paco) [Hur et al. 2013] and companion [Pous 2016], and how we improve on them.

As we reviewed in Section 2, Paco provides incremental reasoning by the parameterized fixed point \( G_f \). It also provides up-to reasoning by combining \( f \) with its greatest respectful closure \( \text{gres}_f \) (i.e., using \( G_{f \circ \text{gres}_f} \)). Pous [2016] shows that the greatest compatible closure \( \text{cpn}_f \), called the \textit{companion}, coincides with \( \text{gres}_f \) and directly admits the incremental and up-to reasoning principles of \( G_{f \circ \text{gres}_f} \). Moreover, the companion admits second-order reasoning, which provides incremental and up-to principles for reasoning about \( \text{clo} \subseteq \text{cpn}_f \).

In our work, we generalize the constructions in two directions. First, we use two parameters to track both the unlocked and guarded knowledge. As briefly discussed in Section 3.2, the companion construction with two parameters \( r \) and \( g \) can be given by \( \text{cpn}_f(r \sqcup f(\text{cpn}_f(r \sqcup g))) \). Second, we parameterize the upper-bound of closures instead of using the greatest compatible/respectful closure. The need for such parameterization was shown in Section 5.4.

\textbf{Distinguishing internal and visible steps} [Sangiorgi and Walker 2001, Exercise 2.4.64] and [Pous 2007] present up-to techniques allowing different up-to closures for internal and visible steps. Among them, [Pous 2007] gives a more formal framework, where two notions of monotonicity (in a more recent terminology, respectfulness) are defined. If a relation \( R \) is \( r \)-simulated (i.e., for internal steps) up-to a monotonic closure and \( v \)-simulated (i.e., for visible steps) up-to a weakly monotonic closure, then \( R \) is contained in the weak (bi)similarity. Notably, up-to weak bisimulation is only weakly monotonic.

Similarly, our work also presents an equational theory for weak bisimulation where internal and visible steps admit different up-to closures. The main challenge we are addressing is to combine such up-to closures with incremental reasoning using four different kinds of knowledge: unlocked/guarded knowledge for internal/visible steps.

\textbf{Other related works} In [Pous 2016], Pous introduced the companion of a function \( f \) by characterizing it as the greatest compatible function for \( f \). Other works have sought more constructivist approaches. Parrow and Weber [2016] give an ordinal-based construction of the companion in classical set theory. Analogously, it turns out that the companion can be obtained in constructive type theory with an inductive tower construction as studied by Schäfer et al. [Schäfer 2019; Smolka et al. 2015].

We have chosen to build our approach on top of paco, but other incremental coinductive techniques exist: incremental
pattern-based coinduction [Popescu and Gunter 2010], circular coinduction [Hausmann et al. 2005], parametric coinduction [Moss 2001]. We refer to Hur et al.’s related work [Hur et al. 2013] for a thorough comparison.

Finally, we introduced through this paper the use of three up-to techniques relevant to our domain of application. Numerous others can be found in Pous [Pous 2016], both derived from the companion and as part of the related work.

References


Yong Kiam Tan, Magnus O. Myreen, Ramana Kumar, Anthony C. J. Fox, Scott Owens, and Michael Norrish. 2016. A new verified compiler backend for CakeML. In ICFP.

A A property about companion

Let \( X = \{(1\epsilon, 2\epsilon)\} \) and \( Y = \{(01\epsilon, 02\epsilon)\} \). We prove that \( \text{cpn}_F(Y) = F(\top) \) for \( F = \text{bisim}_F b_L b_R \text{clo}_\beta \) with arbitrary \( b_L, b_R, \text{clo}_\beta \).

We first define a function \( \text{clo} \) as follows:

\[
\text{clo}(r) = \begin{cases} 
\top & \text{if } X \subseteq r \\
F(\top) & \text{else if } Y \subseteq r \\
\emptyset & \text{otherwise}
\end{cases}
\]

Then \( \text{clo} \) is trivially monotone and compatible as follows. For any \( r \), we show \( \text{clo}(F(r)) \subseteq F(\text{clo}(r)) \) by case analysis on \( r \). First, when \( X \subseteq r \), we have \( \text{clo}(r) = \top \). We also have \( Y \subseteq F(X) \subseteq F(r) \) and \( X \not\subseteq F(r) \) by definition of \( F \). Therefore, we have \( \text{clo}(F(r)) = F(\top) = F(\text{clo}(r)) \). Second, when \( X \not\subseteq r \), we have \( X \not\subseteq F(r) \) and \( Y \not\subseteq F(r) \) by definition of \( F \). Therefore, we have \( \text{clo}(F(r)) = \emptyset \subseteq F(\text{clo}(r)) \).

Now, we have the following inequality:

\[
F(\top) = \text{clo}(Y) \quad \text{(by definition of } \text{clo})
\subseteq \text{cpn}_F(Y) \quad \text{(cpn}_F \text{ includes every compatible func.)}
\subseteq \text{cpn}_F(F(X)) \quad \text{(by definition of } F)
\subseteq F(\text{cpn}_F(X)) \quad \text{(cpn}_F \text{ itself is compatible)}
\subseteq F(\top)
\]

Therefore, we have \( \text{cpn}_F(Y) = F(\top) \).