An Equational Theory for Weak Bisimulation via Generalized Parameterized Coinduction

Yannick Zakowski
University of Pennsylvania
Philadelphia, PA, USA

Paul He
University of Pennsylvania
Philadelphia, PA, USA

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

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.

CCS Concepts • **Software and its engineering → Formal software verification**; • **Theory of computation → Program verification**; *Logic and verification*; *Equational logic and rewriting*.

Keywords Coq, coinduction, up-to techniques, weak bisimulation, equational theory

ACM Reference Format:
Yannick Zakowski, Paul He, Chung-Kil Hur, and Steve Zdancewic. 2020. An Equational Theory for Weak Bisimulation via Generalized Parameterized Coinduction. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP '20), January 20–21, 2020, New Orleans, LA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372885.3373813

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. Firstly, the paco reasoning principles are still too

CPP '20, January 20–21, 2020, New Orleans, LA, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7097-4/20/01.
https://doi.org/10.1145/3372885.3373813

weak, resulting in cumbersome proofs. The limitation is particularly apparent when a proof nests two cofixed points: the inner cofixed point forgets all available accumulated knowledge, leading to redundant reasoning. Secondly, the support for up-to reasoning remains either ad hoc or difficult to manipulate in existing approaches: here we advocate 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 developing modular proof. 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 τ (internal) events. Here we also present our novel proof rules for working with those bisimulations. We also show the problem with working with the companion when trying to define these rules. Section 6 details the implementation of our reasoning principles in Coq. Finally, Section 7 provides a comparison with related work.

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. All of the definitions, metatheory and examples presented here have been verified in Coq. 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 $f \in C \xrightarrow{mon} C$, 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 → C → Prop`.

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

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 $\nu.f$ itself and instead conducting a proof toward some $G_f \in C \xrightarrow{mon} C$ that is parameterized by some accumulated knowledge:

Definition 2.1 (Parameterized greatest fixed point)**.** Define $G \in (C \xrightarrow{mon} C) \xrightarrow{mon} (C \xrightarrow{mon} C)$ to be:

$$G_f\ r \overset{\text{def}}{=} \nu.(\lambda y. f(r \sqcup y))$$

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)**.** $\nu.f \equiv G_f \perp$

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)**.** $G_f\ r \equiv f(r \sqcup G_f\ r)$

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)**.** $y \sqsubseteq G_f\ r \iff y \sqsubseteq G_f\ (r \sqcup y)$

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 aims to prove a goal of the form $y \sqsubseteq \nu.f$. One starts by using Init to obtain $y \sqsubseteq G_f \perp$, 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 (or lazy list, since these streams can also be finite) 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*, τ, and *external* (or *visible*) events $\beta(n)$, terminated (if finite) by the ϵ marker. Here, for simplicity, we assume that visible events carry a natural number. We will sometimes omit the β constructor and just write n (especially in examples) to save space.

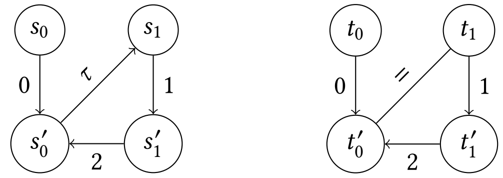
Here are some example streams:

$s_0 =$	01ϵ	finite stream
$s_1 =$	$\tau 0 \tau\tau 1 \epsilon$	finite stream
$s_2 =$	$012 \ldots n(n+1) \ldots$	infinite increasing stream
$s_3 =$	$0\tau 1 \tau 2 \ldots n\tau(n+1) \ldots$	infinite increasing stream
$s_4 =$	$01010101 \ldots$	infinite alternating stream
$s_5 =$	$\tau\tau\tau\tau\tau\tau\tau \ldots$	silent divergence

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 τ 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 τ 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 (which we often abbreviate to eutt). 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 functor, euttF:

$$\texttt{euttF} : \mathcal{P}(\texttt{stream} \times \texttt{stream}) \to \mathcal{P}(\texttt{stream} \times \texttt{stream})$$
$$\approx \; \equiv \nu.\texttt{euttF}$$



$$s_0 = 0\, s_0' \qquad s_0' = \tau\, s_1 \qquad s_1 = 1\, s_1' \qquad s_1' = 2\, s_0'$$
$$t_0 = 0\, t_0' \qquad t_0' = t_1 \qquad t_1 = 1\, t_1' \qquad t_1' = 2\, t_0'$$

Figure 1. Two weakly bisimilar transition systems: illustrating the shortcoming of paco's reasoning principles

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

Lemma 2.5 (euttF Tau Left)**.**

$$X \subseteq \texttt{euttF}(Y) \implies \{(\tau s, t) \mid (s,t) \in X\} \subseteq \texttt{euttF}(Y)$$

Lemma 2.6 (euttF Vis)**.**

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

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 euttF(Y), it suffices to show that s is related to t by euttF(Y)—we can drop a τ 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 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 2. 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.

Let $X_0 = \{(s_0, t_0), (s_1, t_1)\}$ and $X_1 = \{(s'_0, t'_0), (s'_1, t'_1)\}$

$$
\begin{array}{rll}
 & X_0 \subseteq \nu.\mathtt{euttF} \overset{\text{Init}}{\iff} X_0 \subseteq G_{\mathtt{euttF}}\ \emptyset & \\
\overset{\text{Acc }\textbf{(a)}}{\iff} & X_0 \subseteq G_{\mathtt{euttF}}\ X_0 & \\
\overset{\text{Unfold}}{\iff} & X_0 \subseteq \mathtt{euttF}(X_0 \cup G_{\mathtt{euttF}}\ X_0) & \\
\overset{\text{lem. 2.6 }\textbf{(b)}}{\Longleftarrow} & X_1 \subseteq X_0 \cup G_{\mathtt{euttF}}\ X_0 & \\
\Longleftarrow & X_1 \subseteq G_{\mathtt{euttF}}\ X_0 & \\
\overset{\text{Acc }\textbf{(c)}}{\iff} & X_1 \subseteq G_{\mathtt{euttF}}\ (X_0 \cup X_1) & \\
 & \text{We now handle both cases in } X_1 \text{ separately:} & \\
\textbf{rhs:} & (s'_1, t'_1) \in G_{\mathtt{euttF}}\ (X_0 \cup X_1) & \\
\overset{\text{Unfold}}{\iff} & (s'_1, t'_1) \in \mathtt{euttF}(X_0 \cup X_1 \cup G_{\mathtt{euttF}}\ (X_0 \cup X_1)) & \\
\overset{\text{lem. 2.6}}{\Longleftarrow} & (s'_0, t'_0) \in (X_0 \cup X_1 \cup G_{\mathtt{euttF}}\ (X_0 \cup X_1)) & \square \\
\textbf{lhs:} & (s'_0, t'_0) \in G_{\mathtt{euttF}}\ (X_0 \cup X_1) & \\
 & \underline{\text{Solution with redundancy }\textbf{(d)}\text{:}} & \\
\overset{\text{Unfold}}{\iff} & (s'_0, t'_0) \in \mathtt{euttF}(X_0 \cup X_1 \cup G_{\mathtt{euttF}}\ (X_0 \cup X_1)) & \\
\overset{\text{lem. 2.5; 2.6}}{\Longleftarrow} & (s'_1, t'_1) \in (X_0 \cup X_1 \cup G_{\mathtt{euttF}}\ X_0 \cup X_1) & \square \\
 & \underline{\text{Failed attempt without redundancy }\textbf{(e)}\text{:}} & \\
\overset{\text{lem. 2.7}}{\Longleftarrow} & (s_1, t_1) \in G_{\mathtt{euttF}}\ (X_0 \cup X_1) \text{ : we cannot conclude.} &
\end{array}
$$

Figure 2. Shortcoming of paco: an illustrating proof

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 Acc for the second time (marked **(c)**). We had already used Acc once, at point **(a)**, putting X_0 into the accumulated knowledge. Intuitively, this means that after we step under a guard we should be able to use X_0, which is what happens at point **(b)**, 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 **(c)**, 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 Lemmas 2.5 and 2.6 successively. 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.

Performing the case analysis earlier (or proving the equivalence of different states) would have avoided the issue with repeated reasoning in this case. However, this solution is both cumbersome and not always possible. For example, the more complex data type described in Section 6.1 has a branching structure that renders such solutions ineffective.

Intuitively however, we would like to simply ignore this τ 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 τ 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 ($G_{\mathtt{euttF}}$ Tau Left)**.**

$$X \subseteq G_{\mathtt{euttF}}\ (Y) \implies \{(\tau s, t) \mid (s, t) \in X\} \subseteq G_{\mathtt{euttF}}\ (Y)$$

It precisely states that one can strip a τ from the left hand side under a call to $G_{\mathtt{euttF}}$. 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 in Figure 2. 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 $-$.

Definition 3.1 (Generalized parameterized greatest fixed point (first definition))**.**
Define $\hat{G} \in (C \xrightarrow{mon} C) \xrightarrow{mon} (C \xrightarrow{mon} C \xrightarrow{mon} C)$ to be:

$$\hat{G}_f \; r \; g \overset{\text{def}}{=} r \sqcup G_f \; (r \sqcup g)$$

Note that if we pick $r = \bot$, this definition degenerates to $G_f \; g$, 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. [1]

Lemma 3.2 (Init)**.**

$$\hat{G}_f \; \bot \; \bot \equiv G_f \; \bot \equiv \nu.f$$

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

Lemma 3.3 (Final)**.**

$$r \sqcup G_f \; g \sqsubseteq \hat{G}_f \; r \; g$$

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 G .

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)**.**

$$r \sqsubseteq \hat{G}_f \; r \; g$$

Naturally, in order for Base to be sound, the incremental principle extends only the guarded knowledge:

Lemma 3.5 (Acc)**.**

$$x \sqsubseteq \hat{G}_f \; r \; (g \sqcup x) \iff x \sqsubseteq \hat{G}_f \; r \; g$$

Finally, stepping under the guard makes the guarded knowledge available. Note that the pattern of accumulation ensures that we always have the invariant that $r \sqsubseteq g$, which is why erasing r here does not lose information.

Lemma 3.6 (Step)**.**

$$f(\hat{G}_f \; g \; g) \sqsubseteq \hat{G}_f \; r \; g$$

With the addition of the available knowledge parameter to gpaco and its new reasoning principles, we are closer to a more succinct proof for 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 τ without having to continue to go under guards.

$$\begin{array}{rll}
 & X_0 \subseteq \nu.\texttt{euttF} \overset{\textsc{Init}}{\iff} X_0 \subseteq \hat{G}_{\texttt{euttF}} \; \emptyset \; \emptyset & \\
\overset{\textsc{Acc}\ \textbf{(a)}}{\iff} & X_0 \subseteq \hat{G}_{\texttt{euttF}} \; \emptyset \; X_0 & \\
\overset{\textsc{Step}\ \textbf{(b)}}{\Longleftarrow} & X_0 \subseteq \texttt{euttF}(\hat{G}_{\texttt{euttF}} \; X_0 \; X_0) & \\
\overset{\text{lem. } 2.6}{\Longleftarrow} & X_1 \subseteq \hat{G}_{\texttt{euttF}} \; X_0 \; X_0 & \\
\overset{\textsc{Acc}\ \textbf{(c)}}{\iff} & X_1 \subseteq \hat{G}_{\texttt{euttF}} \; X_0 \; (X_0 \cup X_1) & \\
\textbf{rhs:} & (s_1', t_1') \in \hat{G}_{\texttt{euttF}} \; X_0 \; (X_0 \cup X_1) & \\
\overset{\textsc{Step}}{\Longleftarrow} & (s_1', t_1') \in \texttt{euttF}(\hat{G}_{\texttt{euttF}} \; (X_0 \cup X_1) \; (X_0 \cup X_1)) & \\
\overset{\text{lem. } 2.6}{\Longleftarrow} & (s_0', t_0') \in \hat{G}_{\texttt{euttF}} \; (X_0 \cup X_1) \; (X_0 \cup X_1) & \\
\overset{\textsc{Base}}{\Longleftarrow} & (s_0', t_0') \in X_0 \cup X_1 & \square \\
\textbf{lhs:} & (s_0', t_0') \in \hat{G}_{\texttt{euttF}} \; X_0 \; (X_0 \cup X_1) & \\
\overset{\text{lem. } 3.7}{\Longleftarrow} & (s_1, t_1) \in \hat{G}_{\texttt{euttF}} \; X_0 \; (X_0 \cup X_1) & \\
\overset{\textsc{Base}\ \textbf{(d)}}{\Longleftarrow} & (s_1, t_1) \in X_0 & \square
\end{array}$$

Figure 3. Improved proof for Figure 1

Lemma 3.7 ($\hat{G}_{\texttt{euttF}}$ Tau Left, idealized)**.**

$$X \subseteq \hat{G}_{\texttt{euttF}} \; r \; g \implies \{(\tau s, t) \mid (s, t) \in X\} \subseteq \hat{G}_{\texttt{euttF}} \; r \; g$$

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 which eliminates 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_1 is placed under the guard. The payoff comes at point **(d)**, where we can immediately use X_0.

This example shows how the additional parameter allows for smoother reasoning and less redundancy in the proofs. One might wonder: are two parameters enough? Might we need an even more general version with three or four parameters to use in some other proof? The answer is that no, two are sufficient. Intuitively, any particular fact is either available or still guarded. The two parameters partition the knowledge into those categories, and the lemmas manipulate the knowledge precisely.

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.

[1] We overload the lemma names like Init and Acc which are defined both for G_f $-$ and $\hat{G}_f$ $-$ $-$. Which one is meant can easily be distinguished from the context.

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 $clo \in C \to C$ that, given a relation X, extends it to a larger relation $clo(X)$. Then such an up-to technique clo allows us to work with smaller relations when proving, for example, bisimilarity, reducing the effort required in the proof. The power of an up-to technique lies in the fact that the smaller relation X may not be a bisimulation at all. However, for reasoning up-to clo to be sound, X must be contained in a bisimulation. For a more in-depth description of up-to techniques, see [Pous and Sangiorgi 2011].

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

$$\tau_L(R) = \{(\tau^* s, t) \mid (s, t) \in R\}$$

where τ^* means any finite number of τs. Using this up-to technique frees the user from having to manually step through the functor and build the bisimulation relation by manipulating τs one by one on the left side. 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 $clo \in C \xrightarrow{mon} C$ is *compatible* with f if $clo \circ f \sqsubseteq f \circ clo$. Compatible functions are a class of up-to techniques that are nice to work with because they are compositional, so different compatible up-to techniques can be used in a single proof. The companion $\mathsf{cpn}_f \in C \xrightarrow{mon} C$ is the join of all such compatible functions, which is again compatible with f. Then, cpn_f admits nice incremental and up-to principles for coinduction: in particular, $clo(\mathsf{cpn}_f(r)) \sqsubseteq \mathsf{cpn}_f(r)$ for any (not necessarily compatible) function $clo \sqsubseteq \mathsf{cpn}_f$. 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 the base closure $bclo \in C \xrightarrow{mon} C$ as the second argument:

$$\hat{G}_f^{bclo}\ r\ g \stackrel{\text{def}}{=} bclo^*(r \sqcup G_{f \circ bclo^*}\ (r \sqcup g))$$

where $bclo^*$ is the transitive closure of $bclo$.

Note that by choosing the companion as a base closure, we get the equality $\hat{G}_f^{\mathsf{cpn}_f}\ r\ g = \mathsf{cpn}_f(r \sqcup f(\mathsf{cpn}_f(r \sqcup g)))$.

Definition 3.9. We introduce the following useful notation:

$$\bar{G}_f^{bclo}\ g \stackrel{\text{def}}{=} \hat{G}_f^{bclo}\ g\ g$$

Then we can use any up-to function clo bounded by $bclo$, and in fact even larger ones bounded by $\bar{G}_f^{bclo}$.

Lemma 3.10 (Closure)**.** *If $clo \sqsubseteq \bar{G}_f^{bclo}$, then*

$$clo(\hat{G}_f^{bclo}\ r\ g) \sqsubseteq \hat{G}_f^{bclo}\ r\ g$$

Since $bclo \sqsubseteq \bar{G}_f^{bclo}$, in the case $clo = bclo$, it is always valid to use Closure, which will be marked as Closure*.

Using this rule, we can now amend Lemma 3.7: it holds, provided we instantiate $bclo$ with τ_L or another closure that contains it (in the sense of Lemma 3.10). For the overall approach to be sound, the usual criterion required of such a base closure is a notion of compatibility. We work with a relaxed condition, weak compatibility, that can be seen as an instance of a compatible up-to-function function [Pous 2016]:

Definition 3.11 (Weakly compatible closure)**.** $bclo \in C \xrightarrow{mon} C$ is weakly compatible for f if

$$bclo \circ f \sqsubseteq f \circ \bar{G}_f^{bclo}$$

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

Lemma 3.12 (Init)**.** *If $bclo$ is weakly compatible for f, then*

$$\hat{G}_f^{bclo} \perp \perp \sqsubseteq G_f \perp$$

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 τ step, both streams go through intermediate transitions r and r' respectively. Moreover, rather than defining the streams using definitional equality "=", we instead specify them via strong bisimilarity "∼". 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.

$$\begin{array}{cccc} s_0 \sim 0\, s_0' & s_0' \sim r \mathbin{+\!\!+} s_1 & s_1 \sim 1\, s_1' & s_1' \sim 2\, s_0' \\ t_0 \sim 0\, t_0' & t_0' \sim r' \mathbin{+\!\!+} t_1 & t_1 \sim 1\, t_1' & t_1' \sim 2\, t_0' \end{array}$$

Figure 4. Two weakly bisimilar streams when $r \approx r'$

Similarly, when proving equivalence up-to-tau, it is intuitively the case that if $r \approx r'$ and we want to coinductively relate the concatenated streams $r \mathbin{+\!\!+} s \approx r' \mathbin{+\!\!+} t$, 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, up-to prefix and up-to (strong)

$$\frac{bclo\text{ weakly compatible for } f}{\hat{G}_f^{bclo}\ \bot\ \bot \sqsubseteq G_f\ \bot}\ \textsc{Init} \qquad \frac{}{r \sqsubseteq \hat{G}_f^{bclo}\ r\ g}\ \textsc{Base} \qquad \frac{}{r \sqcup G_f\ g \sqsubseteq \hat{G}_f^{bclo}\ r\ g}\ \textsc{Final} \qquad \frac{}{f(\hat{G}_f^{bclo}\ g\ g) \sqsubseteq \hat{G}_f^{bclo}\ r\ g}\ \textsc{Step}$$

$$\frac{x \sqsubseteq \hat{G}_f^{bclo}\ r\ (g \sqcup x)}{x \sqsubseteq \hat{G}_f^{bclo}\ r\ g}\ \textsc{Acc} \qquad \frac{clo \sqsubseteq \bar{G}_f^{bclo}}{clo(\hat{G}_f^{bclo}\ r\ g) \sqsubseteq \hat{G}_f^{bclo}\ r\ g}\ \textsc{Closure} \qquad \frac{}{bclo(\hat{G}_f^{bclo}\ r\ g) \sqsubseteq \hat{G}_f^{bclo}\ r\ g}\ \textsc{Closure}^*$$

Figure 5. Proof rules for generalized parameterized coinduction

bisimilarity:

$$\mathit{prefix}(R) = \{(h_1 +\!\!\!+\, t_1, h_2 +\!\!\!+\, t_2) \mid h_1 \approx h_2 \wedge (t_1, t_2) \in R\}$$

$$\mathit{bisim}(R) = \{(a, b) \mid \exists a', b', a \sim a' \wedge b \sim b' \wedge (a', b') \in R\}$$

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. Note that up-to *prefix* is an instance of the standard up-to context technique [Pous and Sangiorgi 2011].

Using the resulting set of reasoning principles provided by gpaco, 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 of up-to *bisim* and *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 $r_n' +\!\!\!+\, t_1$ 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 gpaco, 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 gpaco. 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 data type 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 data type considered is the same type of potentially finite streams of internal and external events introduced earlier in the paper. Formally, we define $\texttt{stream} \overset{\text{def}}{=} \nu.\texttt{streamF}$ where:

$$\begin{aligned}\texttt{streamF}\ X \overset{\text{def}}{=} \{\epsilon\} &\cup \{\tau \cdot s \mid s \in X\} \\ &\cup \{\beta(n) \cdot s \mid s \in X,\ n \in \mathbb{N}\}\end{aligned}$$

An element of the resulting type stream is hence a potentially infinite trace consisting of internal steps, represented as τ constructors, and visible events, emitting natural numbers, represented as β constructors. Such a data type can for instance be thought of as the observable trace of an interactive program’s execution.

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

Defining a concatenation operation over streams, concat, is straightforward: let $\texttt{concat} \overset{\text{def}}{=} \nu.\texttt{concatF}$ where

$$\begin{aligned}\texttt{concatF}\ \mathit{concat\_} \overset{\text{def}}{=} \lambda s\ k.\ &\texttt{case}\ s\ \texttt{of} \\ &\mid \epsilon \Rightarrow k \\ &\mid \tau \cdot s \Rightarrow \tau \cdot (\mathit{concat\_}\ s\ k) \\ &\mid \beta(n) \cdot s \Rightarrow \beta(n) \cdot (\mathit{concat\_}\ s\ k)\end{aligned}$$

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 coinductive 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

$$
\begin{aligned}
&\mathtt{fix}\ \mathtt{bisimF}\ (b_L\ b_R : \mathtt{bool})\ clo_\beta\ X \overset{\text{def}}{=}\\
&\qquad \{(\epsilon, \epsilon)\} \cup\\
&\qquad \{(\tau \cdot s, \tau \cdot t) \ \mid\ (s, t) \in X\} \cup\\
&\qquad \{(\beta(n) \cdot s, \beta(n) \cdot t) \ \mid\ (s, t) \in clo_\beta(X),\ n \in \mathbb{N}\} \cup\\
&\qquad \{(\tau \cdot s, t) \ \mid\ b_L = \mathtt{true} \land (s, t) \in \mathtt{bisimF}\ b_L\ b_R\ clo_\beta\ X\} \cup\\
&\qquad \{(s, \tau \cdot t) \ \mid\ b_R = \mathtt{true} \land (s, t) \in \mathtt{bisimF}\ b_L\ b_R\ clo_\beta\ X\}\\
&\mathtt{bisim}\ b_L\ b_R \overset{\text{def}}{=}\ G_{\mathtt{bisimF}\ b_L\ b_R\ \mathtt{id}}\ \bot
\end{aligned}
$$

Figure 6. Definition of a family of bisimulations over streams

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 τ 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 `bisim`, a family of relations over streams. Let us for now ignore its three parameters and focus at a high level on the functor `bisimF _ _ _ X`. We use the `fix` keyword as a notation to express `bisimF` itself is defined as a smallest fixed point.

There are five ways we may relate two streams: 1. by matching ϵ constructs, 2. by matching τ and co-recursing, 3. by matching identical β and co-recursing, 4. by stripping a τ from the left and recursing or 5. by stripping a τ from the right and recursing. Note the use of a *recursive* call when stripping τ in the asymmetric cases (4) and (5): if we were to iterate co-recursively, then an infinite co-recursive chain of application of rule (4) would relate the silently diverging stream to any stream.

The three parameters to `bisimF` 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 clo_β parameter, of type $\mathcal{P}(stream \times stream) \rightarrow \mathcal{P}(stream \times stream)$ 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 apply clo_β to it.

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 clo_β parameter to the identity closure `id` in order to define the high level relations we are interested in. It is straightforward to check that `bisimF` b_L b_R clo_β is monotone for any monotone clo_β, in particular for `id`. We therefore can define the greatest fixed point `bisim` b_L b_R using `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)**.**

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

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

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

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

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 τ:

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

$$s \gtrsim t \overset{\text{def}}{=} \mathtt{bisim}\ \mathtt{true}\ \mathtt{false}\ s\ t$$

Notice the following subrelation inclusions: $\sim\ \subseteq\ \gtrsim\ \subseteq\ \approx$.

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 bisimilarity only allows us to step through the functor `bisimF`, forcing us systematically to nest an induction to account for possible bounded stripping of τs, 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).

$$\frac{(s_1, s_1') \in \texttt{bisim}\ b_L\ b_R \quad (s_1', s_2') \in r \quad (s_2, s_2') \in \texttt{bisim}\ b_L'\ b_R'}{(s_1, s_2) \in \texttt{bisim\_trans\_clo}\ b_L\ b_R\ b_L'\ b_R'\ r}$$

Each pair of flags defines the instances of `bisim` 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).

$$\mathcal{U} \overset{\text{def}}{=} \texttt{bisim\_trans\_clo true true true true}$$

Let us emphasize why such arbitrary, *undirected*, up-to-tau rewriting provided by $\mathcal{U}$ is an unsound principle in general, which was first shown by Sangiorgi and Milner [1992]. 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, $\mathcal{U}$ could hence be misused to *introduce* a τ constructor that could then be used as a guard, allowing for unsound circular reasoning. To illustrate the problem concretely, let us assume for a moment that the precondition of the Closure principle from Figure 5 is available for $\mathcal{U}$. The following proof would then be valid:

$$\begin{aligned}
0\epsilon \approx 1\epsilon \ &\overset{\textsc{Init}}{\Longleftrightarrow}\ (0\epsilon, 1\epsilon) \in \hat{G}_{\texttt{euttF}}\ \emptyset\ \emptyset \\
\overset{\textsc{Acc}}{\Longleftrightarrow}\ &(0\epsilon, 1\epsilon) \in \hat{G}_{\texttt{euttF}}\ \emptyset\ \{(0\epsilon, 1\epsilon)\} \\
\overset{\textsc{Closure}(\mathcal{U})}{\Longleftarrow}\ &(\tau 0\epsilon, \tau 1\epsilon) \in \hat{G}_{\texttt{euttF}}\ \emptyset\ \{(0\epsilon, 1\epsilon)\} \\
\overset{\text{Step}}{\Longleftarrow}\ &(0\epsilon, 1\epsilon) \in \hat{G}_{\texttt{euttF}}\ \{(0\epsilon, 1\epsilon)\}\ \{(0\epsilon, 1\epsilon)\} \\
\overset{\text{Base}}{\Longleftarrow}\ &(0\epsilon, 1\epsilon) \in \{(0\epsilon, 1\epsilon)\} \quad \square
\end{aligned}$$

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

We will come back to $\mathcal{U}$ 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).

$$\mathcal{D} \overset{\text{def}}{=} \texttt{bisim\_trans\_clo true false true false}$$

The $\mathcal{D}$ 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 *no more* τs than the previous one. It is intuitively clear that this substitution is always sound since it cannot introduce a guard. Note that this is the up-to expansion technique presented by Sangiorgi and Milner [1992] to solve the problem of up-to weak bisimularity above. 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 $\mathcal{D}$ provides a sound up-to reasoning principle with respect to $\approx$. This soundness holds in the sense that $\mathcal{D}$ satisfies the precondition from Lemma 3.12 with respect to the functor `euttF` $\overset{\text{def}}{=}$ `bisimF true 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 $\mathcal{D}$ as the base closure.

Lemma 4.7 (Initialization for $\mathcal{D}$ with respect to `euttF`). *For any monotone clo_β such that $\mathcal{D} \circ clo_\beta \subseteq clo_\beta \circ \mathcal{D}$, $\mathcal{D}$ is weakly compatible for* `euttF` *clo_β.*

We can at this stage already establish a certain number of facts about our instances of `bisim`. By picking in particular clo_β = `id`, the closure used in the definition of `euttF`, we can derive the following reasoning principle by applying Closure*.

Theorem 4.8 ($\approx$ is a congruence for $\gtrsim$).

$$\frac{s' \gtrsim s \quad s' \approx t' \quad t' \gtrsim t}{s \approx t}$$

We then prove that `bisim` defines equivalence relations:

Lemma 4.9. *$\sim$ and $\approx$ are equivalence relations. $\gtrsim$ is reflexive and transitive.*

And finally show that `bisim` $b_L\ b_R$ 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).

$$\frac{h_1 \approx h_2 \quad (t_1, t_2) \in r}{(h_1 + t_1, h_2 + t_2) \in C\ r}$$

The soundness of the closure is embodied by showing that Lemma 3.10 can be instantiated for C with respect to `euttF`, with $\mathcal{D}$ for the base closure:

Lemma 4.11 (Compatibility of C with respect to `euttF`). *For any clo_β monotone such that $C \circ clo_\beta \subseteq clo_\beta \circ C$ and* `id` *$\subseteq clo_\beta$, we have $C \subseteq \bar{G}^{\mathcal{D}}_{\texttt{euttF}\ clo_\beta}$.*

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

Theorem 4.12 ($\sim$ is a congruence for `concat`)**.**

$$\frac{h_1 \sim h_2 \quad t_1 \sim t_2}{h_1 +\!\!+\, t_1 \sim h_2 +\!\!+\, t_2}$$

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 ((`stream`,$+\!\!+$) forms a monoid)**.**

$$\epsilon +\!\!+\, s \sim s \qquad s +\!\!+\, \epsilon \sim s \qquad (r +\!\!+\, s) +\!\!+\, t \sim r +\!\!+\, (s +\!\!+\, t)$$

5 An Equational Theory for Weak Bisimulations

Section 4 introduced the `stream` data type 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, $\mathcal{D}$, 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 β, but not when guarded by a τ.

In order to alleviate these difficulties, we abstract away from the low-level use of `gpaco` and define in 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` $r_\beta\ r_\tau\ g_\beta\ g_\tau$. Before getting to its formal definition, we sketch the intuition it carries. The relation takes four parameters, each of type $\mathcal{P}$(`stream`$\times$ `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 guards 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 τ steps, but we remain certain that a cycle is found if we get back to a state reached under a β step. As such, β guards are stronger than τ guards when reasoning up-to-tau.

The main tool we will use to enable more reasoning principles under β guards than τ guards is the clo_β argument introduced in the definition of `bisim`, Figure 6, and which has been left unexploited through Section 4. Recall that this parameter is a closure up-to which is applied to the co-recursive call under a β constructor. The closure we consider is defined as follows:

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

$$\mathcal{V}_{g_\beta}\ r \overset{\text{def}}{=} \bar{G}^{\mathcal{D}}_{\texttt{euttF id}}\ \mathcal{U}(r \cup g_\beta).$$

The closure $\mathcal{V}_{g_\beta}$ is best understood right to left. At its core, it simply extends the relation r with the β guarded knowledge g_β. Since it will only be accessible under β guards, it is also sound to close this knowledge up to *undirected* transitivity, $\mathcal{U}$, to allow for arbitrary rewriting by weak bisimilarity. Finally, by definition of `bisimF`, using $\mathcal{V}_{g_\beta}$ in place of the clo_β argument permits its use right as we strip off a pair of β 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 `gpaco` is a convenient way to make this possible.

We now turn to the definition of `euttG` itself:

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

$$\texttt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau \overset{\text{def}}{=} \hat{G}^{\mathcal{D}}_{\texttt{euttF}\ (\mathcal{V}_{g_\beta})}\ (\mathcal{U}(r_\beta) \cup r_\tau))\ g_\tau$$

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

Having motivated the definition of `euttG` by the intuitive reasoning principles it should satisfy, we formalize these principles in the following subsection.

Soundness

$$\frac{(s,t) \in \mathtt{euttG}\ \emptyset\ \emptyset\ \emptyset\ \emptyset}{s \approx t}\ \textsc{Init} \qquad \frac{s \approx t}{(s,t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau}\ \textsc{Final}$$

Knowledge manipulation

$$\frac{(s,t) \in r_\beta \cup r_\tau}{(s,t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau}\ \textsc{Base} \qquad \frac{x \subseteq \mathtt{euttG}\ r_\beta\ r_\tau\ (g_\beta \cup x)\ (g_\tau \cup x)}{x \subseteq \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau}\ \textsc{Acc}$$

Stream processing

$$\frac{}{(\epsilon,\epsilon) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau}\ \textsc{Ret} \qquad \frac{(s,t) \in \mathtt{euttG}\ r_\beta\ g_\tau\ g_\beta\ g_\tau}{(\tau \cdot s, \tau \cdot t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau}\ \tau\_\textsc{Step} \qquad \frac{(s,t) \in \mathtt{euttG}\ g_\beta\ g_\beta\ g_\beta\ g_\beta}{(\beta(n) \cdot s, \beta(n) \cdot t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau}\ \beta\_\textsc{Step}$$

Up to reasoning

$$\frac{(s,t) \in \mathcal{D}(\mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau)}{(s,t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau}\ \textsc{TransD} \qquad \frac{(s,t) \in \mathcal{U}(\mathtt{euttG}\ r_\beta\ r_\beta\ g_\beta\ r_\beta)}{(s,t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau}\ \textsc{TransU} \qquad \frac{(s,t) \in C(\mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau)}{(s,t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau}\ \textsc{ConcatC}$$

Figure 7. Equational theory for parameterized equivalence up-to-tau. $\mathcal{D}$, $\mathcal{U}$ and C are the closures for which up-to reasoning is possible: directed and undirected transitivity, and concatenation.

5.2 An Equational Theory for `euttG`

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 `euttG`. The *knowledge manipulation* rules provide the core coinductive principles specialized to weak bisimulation. The *stream processing* rules give specialized principles to step under `euttF` constructors. Finally, we provide support for three *up-to* reasoning principles. All rules maintain the following implicit invariant for `euttG`: $r_\beta \subseteq r_\tau \subseteq g_\tau \subseteq g_\beta$.

Soundness The relation between `euttG` and $\approx$ is similar to the one between paco and 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 `euttG` realm by assuming no initial knowledge.

Theorem 5.3 (Init).

$$(s,t) \in \mathtt{euttG}\ \emptyset\ \emptyset\ \emptyset\ \emptyset \implies s \approx t$$

Using Init, we can hence start a `euttG`-based proof. Conversely, since `euttG` 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, `euttG` always contains $\approx$.

Theorem 5.4 (Final).

$$s \approx t \implies (s,t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau$$

Knowledge manipulation The `euttG` 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 gpaco counterpart by giving access to all unlocked knowledge.

Theorem 5.5 (Base).

$$(s,t) \in r_\beta \cup r_\tau \implies (s,t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau$$

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 `euttG`, one can extend the guarded knowledge by assuming that x is contained in this knowledge:

Theorem 5.6 (Acc).

$$x \subseteq \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau \iff x \subseteq \mathtt{euttG}\ r_\beta\ r_\tau\ (g_\beta \cup x)\ (g_\tau \cup x)$$

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).

$$(\epsilon,\epsilon) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau$$

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

Theorem 5.8 (τ step).

$$(t,s) \in \mathtt{euttG}\ r_\beta\ g_\tau\ g_\beta\ g_\tau \implies (\tau \cdot s, \tau \cdot t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau$$

Finally, visible steps propagate the guarded knowledge to all parameters.

Theorem 5.9 (β step).

$$\begin{aligned} &(t,s) \in \mathtt{euttG}\ g_\beta\ g_\beta\ g_\beta\ g_\beta \\ &\quad \implies (\beta(n) \cdot s, \beta(n) \cdot t) \in \mathtt{euttG}\ r_\beta\ r_\tau\ g_\beta\ g_\tau \end{aligned}$$

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 transD and concatC: one can simply make a call to the corresponding closure at any time.

$$
\begin{aligned}
& X_0 \subseteq \nu.\texttt{euttF} \overset{\textsc{Init}}{\Longleftarrow} X_0 \subseteq \texttt{euttG}\,\emptyset\,\emptyset\,\emptyset\,\emptyset \\
\overset{\textsc{Acc}}{\Longleftarrow}\ & X_0 \subseteq \texttt{euttG}\,\emptyset\,\emptyset\,X_0\,X_0 \\
\overset{\textsc{TransU}}{\Longleftarrow}\ & \{(0\,s_0', 0\,t_0'), (1\,s_1', 1\,t_1')\} \subseteq \texttt{euttG}\,\emptyset\,\emptyset\,X_0\,\emptyset \\
\overset{\beta\textsc{\_Step}}{\Longleftarrow}\ & X_1 \subseteq \texttt{euttG}\,X_0\,X_0\,X_0\,X_0 \\
\overset{\textsc{Acc}}{\Longleftarrow}\ & X_1 \subseteq \texttt{euttG}\,X_0\,X_0\,(X_0 \cup X_1)\,(X_0 \cup X_1) \\
\textbf{lhs:}\ & (s_0', t_0') \in \texttt{euttG}\,X_0\,X_0\,(X_0 \cup X_1)\,(X_0 \cup X_1) \\
\overset{\textsc{TransU}}{\Longleftarrow}\ & (r +\!\!\!+\, s_1, r' +\!\!\!+\, t_1) \in \texttt{euttG}\,X_0\,X_0\,(X_0 \cup X_1)\,X_0 \\
\overset{\textsc{ConcatC}}{\Longleftarrow}\ & (s_1, t_1) \in \texttt{euttG}\,X_0\,X_0\,(X_0 \cup X_1)\,X_0 \\
\overset{\textsc{Base}}{\Longleftarrow}\ & (s_1, t_1) \in X_0 \qquad \square \\
\textbf{rhs:}\ & (s_1', t_1') \in \texttt{euttG}\,X_0\,X_0\,(X_0 \cup X_1)\,(X_0 \cup X_1) \\
\overset{\textsc{TransU}}{\Longleftarrow}\ & (2\,s_0', 2\,t_0') \in \texttt{euttG}\,X_0\,X_0\,(X_0 \cup X_1)\,X_0 \\
\overset{\beta\textsc{\_Step}}{\Longleftarrow}\ & (s_0', t_0') \in \texttt{euttG}\,(X_0{\cup}X_1)\,(X_0{\cup}X_1)\,(X_0{\cup}X_1)\,(X_0{\cup}X_1) \\
\overset{\textsc{Base}}{\Longleftarrow}\ & (s_0', t_0') \in X_0 \cup X_1 \qquad \square
\end{aligned}
$$

Figure 8. Practical use of euttG: a proof example

Theorem 5.10 (Directed transitive closure).

$$(s, t) \in \mathcal{D}(\texttt{euttG}\,r_\beta\,r_\tau\,g_\beta\,g_\tau) \implies (s, t) \in \texttt{euttG}\,r_\beta\,r_\tau\,g_\beta\,g_\tau$$

Theorem 5.11 (Concat closure).

$$(s, t) \in \mathcal{C}(\texttt{euttG}\,r_\beta\,r_\tau\,g_\beta\,g_\tau) \implies (s, t) \in \texttt{euttG}\,r_\beta\,r_\tau\,g_\beta\,g_\tau$$

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

Theorem 5.12 (Undirected transitive closure).

$$(s, t) \in \mathcal{U}(\texttt{euttG}\,r_\beta\,r_\beta\,g_\beta\,r_\beta) \implies (s, t) \in \texttt{euttG}\,r_\beta\,r_\tau\,g_\beta\,g_\tau$$

We now illustrate a use of this interface.

5.3 Practical Use of euttG

Consider the following two streams:

$$
\begin{array}{llll}
s_0 \approx 0\,s_0' & s_0' \approx r +\!\!\!+\, s_1 & s_1 \approx 1\,s_1' & s_1' \approx 2\,s_0' \\
t_0 \approx 0\,t_0' & t_0' \approx r' +\!\!\!+\, t_1 & t_1 \approx 1\,t_1' & t_1' \approx 2\,t_0'
\end{array}
$$

This example differs from Figure 4 in that each of the states are related to one another by *weak* bisimilarity. To prove that $s_0 \approx t_0$ and $s_1 \approx t_1$, the same proof as before using just gpaco will not work, since we need to use $\mathcal{U}$, a context-sensitive closure. However, the proof remains straightforward using euttG, assuming still that we know $r \approx r'$, as depicted in Figure 8.

Notice in particular how TransU allows us to rewrite up-to-tau equations, at the cost each time of losing the knowledge locked behind a τ guard.

5.4 Essential Need for the Base Closure

We show that the companion closure is inconsistent with the rules of euttG, so that it cannot be used as a base closure.

$$
\begin{aligned}
& X \subseteq \nu.F \overset{\textsc{Init}}{\Longleftarrow} X \subseteq \texttt{euttG}\,\emptyset\,\emptyset\,\emptyset\,\emptyset \\
\overset{\textsc{Acc}}{\Longleftarrow}\ & X \subseteq \texttt{euttG}\,\emptyset\,\emptyset\,X\,X \\
\overset{\textsc{TransU}}{\Longleftarrow}\ & X \subseteq \mathcal{U}(\texttt{euttG}\,\emptyset\,\emptyset\,X\,\emptyset) \\
\overset{\text{by (1)}}{\Longleftarrow}\ & X \subseteq \mathcal{U}(\texttt{cpn}_F(\texttt{euttG}\,\emptyset\,\emptyset\,X\,\emptyset)) \\
\overset{\text{by (3)}}{\Longleftarrow}\ & X \subseteq \mathcal{U}(\texttt{cpn}_F(Y)) \\
\overset{\text{by (2)}}{\Longleftrightarrow}\ & X \subseteq \mathcal{U}(F(\top)) \\
\Longleftarrow\ & X \subseteq \mathcal{U}(F(X))\ (\text{since } (\tau 1\epsilon, \tau 2\epsilon) \in F(X)) \qquad \square
\end{aligned}
$$

Figure 9. A contradiction when the companion is used as the base closure

To this end, for any definition of euttG satisfying the rules in Figure 7, suppose that it is closed under the companion $\texttt{cpn}_F$ for $F = \texttt{bisimF}\ b_L\ b_R\ clo_\beta$ with arbitrary b_L, b_R, clo_β:

$$\texttt{cpn}_F(\texttt{euttG}\,r_\beta\,r_\tau\,g_\beta\,g_\tau) \subseteq \texttt{euttG}\,r_\beta\,r_\tau\,g_\beta\,g_\tau \tag{1}$$

Let $X = \{(1\epsilon, 2\epsilon)\}$ and $Y = \{(01\epsilon, 02\epsilon)\}$. For $\top : \texttt{stream} \times \texttt{stream}$, we have:

$$\texttt{cpn}_F(Y) = F(\top) \tag{2}$$

$$Y \subseteq \texttt{euttG}\,\emptyset\,\emptyset\,X\,\emptyset \tag{3}$$

The proof of (2) is given in Appendix A.1. (3) follows by applying β_Step then Base.

Then, as shown in Figure 9, we can derive a contradiction, that $1\epsilon \approx 2\epsilon$. The root of the issue is that the companion construction contains non-structural "junk" when provided a false assumption like Y above. Where we would want $\texttt{cpn}_F(Y)$ to contain exactly the pairs of streams equivalent modulo Y, it also ends up containing nonsensical pairs such as $(\tau 1\epsilon, \tau 2\epsilon)$.

6 Implementation in the Coq Proof Assistant and Large Scale 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 https://github.com/snu-sf/paco.

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 https://github.com/DeepSpec/InteractionTrees/. In particular, the equational theory is developed in the `/theories/Eq` directory.

7 Discussion and Related Work

Paco and Companion We start by discussing how our contribution builds on existing works, namely parameterized coinduction (Paco) [Hur et al. 2013] and the 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 $\mathtt{gres}_f$ (*i.e.*, using $G_{f \circ \mathtt{gres}_f}$). Pous [2016] shows that the greatest compatible closure $\mathtt{cpn}_f$, called the *companion*, coincides with $\mathtt{gres}_f$ and directly admits the incremental and up-to reasoning principles of $G_{f \circ \mathtt{gres}_f}$. Moreover, the companion admits second-order reasoning, which provides incremental and up-to principles for reasoning about $clo \sqsubseteq \mathtt{cpn}_f$.

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 $\mathtt{cpn}_f(r \sqcup f(\mathtt{cpn}_f(r \sqcup g)))$. 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.

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 τ-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.

Aristizabal et al. [2016] have developed a general framework to reason about notions of weak steps vs. strong steps (passive vs. active in their terminology) when establishing a bisimulation. Simulations can generally be phrased in term of a relation $\mathcal{R}$ that progresses to itself: $\mathcal{R} \rightarrowtail \mathcal{R}$. Under this formulation, an up-to technique is a function f on relations such that when $\mathcal{R} \rightarrowtail f(\mathcal{R})$, then $\mathcal{R}$ is included in the bisimilarity relation. In order to account for a distinction of the stepping relation between a passive part and an active part, they introduce the notion of diacritical progress: $\mathcal{R} \twoheadrightarrow \mathcal{Q}, \mathcal{S}$ expresses that $\mathcal{R}$ progresses toward $\mathcal{Q}$ in the passive case, toward $\mathcal{S}$ in the active case. With this tool, an up-to technique in the usual sense (called strong) is a function f such that $\mathcal{R} \twoheadrightarrow f(\mathcal{R}), f(\mathcal{R})$ implies that $\mathcal{R}$ is in the bisimilarity relation. This definition also extends to functions f such that $\mathcal{R} \twoheadrightarrow \mathcal{R}, f(\mathcal{R})$ implies the same. These up-to techniques make explicit the fact that up-to reasoning is only enabled when performing active steps. In [Aristizabal et al. 2016], they develop sufficient conditions for using strong and regular up-to techniques in terms of the notions of evolution and compatibility of functions, adapted to the diacritical setting. [Biernacki et al. 2019] goes further by generalizing this view to the lattice-theoretic setting. This generalization allows them to introduce a notion of diacritical companion defined as the greatest diacritically compatible function, extending on both their and Pous' work.

This approach, whose contribution is orthogonal to that of this paper, we conjecture could be defined in `gpaco`. The development of `euttG`, and of the soundness of the `transU` rule in particular, might then fit nicely into this framework, potentially benefiting from this more principled approach in being easier to define. Investigating this conjecture formally would be an interesting approach for future work.

Other Related Works In [Pous 2016], Pous introduced the companion of a function f by characterizing it as the greatest compatible function for f. Parrow and Weber [2016] give a more explicit, 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].

[Danielsson 2017] presents a class of up-to techniques using size-preserving functions, which use sized types to prove the soundness of the techniques. This class of techniques is shown to be related to Pous' companion, but does not include some useful up-to techniques. Namely, Danielsson shows that techniques related to transitivity, such as those discussed in this paper, do not easily fit into the framework of size-preserving functions.

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.

A Appendix

A.1 A Property about the Companion

Let $X = \{(1\epsilon, 2\epsilon)\}$ and $Y = \{(01\epsilon, 02\epsilon)\}$. We prove that $\mathsf{cpn}_F(Y) = F(\top)$ for $F = \mathtt{bisimF}\ b_L\ b_R\ clo_\beta$ with arbitrary b_L, b_R, clo_β.

We first define a function clo as follows:

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

Then clo is trivially monotone and compatible as follows. For any r, we show $clo(F(r)) \subseteq F(clo(r))$ by case analysis on r. First, when $X \subseteq r$, we have $clo(r) = \top$. We also have $Y \subseteq F(X) \subseteq F(r)$ and $X \nsubseteq F(r)$ by definition of F. Therefore, we have $clo(F(r)) = F(\top) = F(clo(r))$. Second, when $X \nsubseteq r$, we have $X \nsubseteq F(r)$ and $Y \nsubseteq F(r)$ by definition of F. Therefore, we have $clo(F(r)) = \emptyset \subseteq F(clo(r))$.

Now, we have the following inequality:

$$\begin{aligned} F(\top) &= clo(Y) && \text{(by definition of } clo\text{)} \\ &\subseteq \mathsf{cpn}_F(Y) && (\mathsf{cpn}_F \text{ includes every compatible func.)} \\ &\subseteq \mathsf{cpn}_F(F(X)) && \text{(by definition of } F\text{)} \\ &\subseteq F(\mathsf{cpn}_F(X)) && (\mathsf{cpn}_F \text{ itself is compatible)} \\ &\subseteq F(\top) \end{aligned}$$

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

Acknowledgments

This work was funded by the National Science Foundation's Expedition in Computing *The Science of Deep Specification* under award 1521539 (Weirich, Zdancewic, Pierce) with additional support by the ONR grant *REVOLVER* award N00014-17-1-2930, and by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (2017R1A2B2007512). We are grateful to all the members of the DeepSpec project for their collaboration and feedback, and we greatly appreciate the reviewers' comments and suggestions.

References

Andres Aristizabal, Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. 2016. Environmental Bisimulations for Delimited-Control Operators with Dynamic Prompt Generation. *Logical Methods in Computer Science* 13 (11 2016). https://doi.org/10.23638/LMCS-13(3:27)2017

Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. 2019. Diacritical Companions. In *MFPS 2019-Mathematical Foundations of Programming Semantics XXXV*. London, United Kingdom. https://doi.org/10.1016/j.entcs.2019.09.003

Nils Anders Danielsson. 2017. Up-to Techniques Using Sized Types. *Proc. ACM Program. Lang.* 2, POPL, Article 43 (Dec. 2017), 28 pages. https://doi.org/10.1145/3158131

Daniel Hausmann, Till Mossakowski, and Lutz Schröder. 2005. Iterative Circular Coinduction for CoCasl in Isabelle/HOL. In *Fundamental Approaches to Software Engineering*, Maura Cerioli (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 341–356.

Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. 2013. The Power of Parameterization in Coinductive Proof. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '13)*. ACM, New York, NY, USA, 193–206. https://doi.org/10.1145/2429069.2429093

Xavier Leroy. 2009. Formal verification of a realistic compiler. *Commun. ACM* 52, 7 (2009), 107–115. https://doi.org/10.1145/1538788.1538814

Thomas Letan, Yann Régis-Gianas, Pierre Chifflier, and Guillaume Hiet. 2018. Modular Verification of Programs with Effects and Effect Handlers in Coq. In *Formal Methods - 22nd International Symposium, FM 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 15-17, 2018, Proceedings*. 338–354. https://doi.org/10.1007/978-3-319-95582-7_20

Lawrence S. Moss. 2001. Parametric Corecursion. *Theor. Comput. Sci.* 260, 1-2 (June 2001), 139–163. https://doi.org/10.1016/S0304-3975(00)00126-2

Joachim Parrow and Tjark Weber. 2016. The Largest Respectful Function. *Logical Methods in Computer Science* Volume 12, Issue 2 (June 2016). https://doi.org/10.2168/LMCS-12(2:11)2016

Andrei Popescu and Elsa L. Gunter. 2010. Incremental Pattern-based Coinduction for Process Algebra and Its Isabelle Formalization. In *Proceedings of the 13th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'10)*. Springer-Verlag, Berlin, Heidelberg, 109–127. https://doi.org/10.1007/978-3-642-12032-9_9

Damien Pous. 2007. New up-to techniques for weak bisimulation. *Theoretical Computer Science* 380, 1 (2007), 164 – 180. https://doi.org/10.1016/j.tcs.2007.02.060 Automata, Languages and Programming.

Damien Pous. 2016. Coinduction All the Way Up. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '16)*. ACM, New York, NY, USA, 307–316. https://doi.org/10.1145/2933575.2934564

Damien Pous and Davide Sangiorgi. 2011. *Enhancements of the bisimulation proof method.* Cambridge University Press, 233–289. https://doi.org/10.1017/CBO9780511792588.007

Davide Sangiorgi and Robin Milner. 1992. The Problem of "Weak Bisimulation Up to". In *Proceedings of the Third International Conference on Concurrency Theory (CONCUR '92)*. Springer-Verlag, London, UK, UK, 32–46. http://dl.acm.org/citation.cfm?id=646727.703207

Davide Sangiorgi and David Walker. 2001. *PI-Calculus: A Theory of Mobile Processes.* Cambridge University Press, New York, NY, USA.

Steven Schäfer. 2019. *Engineering Formal Systems in Constructive Type Theory.* Ph.D. Dissertation. Saarland University.

Gert Smolka, Steven Schäfer, and Christian Doczkal. 2015. Transfinite Constructions in Classical Type Theory. In *Interactive Theorem Proving*, Christian Urban and Xingyuan Zhang (Eds.). Springer International Publishing, Cham, 391–404.

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*.

Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. 2020. Interaction Trees. In *Proceedings of the 47th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '20)*. ACM, New York, NY, USA.