Coinductive Proofs for Temporal Hyperliveness

ARTHUR CORRENSON, CISPA Helmholtz Center for Information Security, Germany
BERND FINKBEINER, CISPA Helmholtz Center for Information Security, Germany

Temporal logics for hyperproperties have recently emerged as an expressive specification technique for relational properties of reactive systems. While the model checking problem for such logics has been widely studied, there is a scarcity of deductive proof systems for temporal hyperproperties. In particular, hyperproperties with an alternation of universal and existential quantification over system executions are rarely supported. In this paper, we focus on hyperproperties of the form $\forall^*\exists^*\psi$, where ψ is a safety relation. We show that hyperproperties of this class – which includes many hyperliveness properties of interest – can always be approximated by coinductive relations. This enables intuitive proofs by coinduction. Based on this observation, we define HyCo (**Hy**perproperties, **Co**inductively), a mechanized framework to reason about temporal hyperproperties within the Coq proof assistant. We detail the construction of HyCo, provide a proof of its soundness, and exemplify its use by applying it to the verification of reactive systems modeled as imperative programs with nondeterminism and I/O.

CCS Concepts: • **Theory of computation → Modal and temporal logics**; *Program verification*; **Logic and verification**.

Additional Key Words and Phrases: Coinduction, Temporal Hyperproperties, Coq

ACM Reference Format:
Arthur Correnson and Bernd Finkbeiner. 2025. Coinductive Proofs for Temporal Hyperliveness. *Proc. ACM Program. Lang.* 9, POPL, Article 53 (January 2025), 28 pages. https://doi.org/10.1145/3704889

1 Introduction

Temporal logics for hyperproperties like HyperLTL [16] combine temporal reasoning over infinite executions with the ability to quantify universally and existentially over multiple executions. Such logics are ideally suited to verify reactive systems, where the continuous interaction of the system with its environment results in an infinite sequence of events. In this context, temporal logics for hyperproperties enable the specification of information-flow security policies as well as other relational properties like robustness, knowledge, and causality. For example, the information-flow policy of *noninference* [32] requires that an observer of a system should not be able to observe any difference in the behavior of the system if we replace its high-security inputs by a dummy value λ. Thus, from the point of view of a potential attacker, sensitive data manipulated by the system are indistinguishable from λ. Expressing this requirement formally requires quantification over several executions of the system. In HyperLTL, noninference is expressed as the formula

$$\forall\tau_1\,\exists\tau_2\,\Box(hi_{\tau_2} = \lambda \land lo_{\tau_1} = lo_{\tau_2})$$

where hi denotes the high-security inputs and lo the low-security outputs of the system. This formula requires for every execution the existence of another execution where the high-security

Authors' Contact Information: Arthur Correnson, CISPA Helmholtz Center for Information Security, Saarbruecken, Germany, arthur.correnson@cispa.de; Bernd Finkbeiner, CISPA Helmholtz Center for Information Security, Saarbruecken, Germany, finkbeiner@cispa.de.

ACM 2475-1421/2025/1-ART53
https://doi.org/10.1145/3704889

inputs to the system are replaced by a dummy value λ and yet the low-security outputs remain the same. Noninference is an example of *hyperliveness* property [17].

There has been a lot of recent research on finite-state and abstraction-based model checking for temporal hyperproperities [10, 12, 13, 26, 28]. Somewhat surprisingly, however, there is a scarcity of deductive proof systems supporting logics like HyperLTL. In particular, contrary to more traditional Hoare-style program logics, there exists, to the best of our knowledge, no mechanized framework to reason about temporal hyperproperties within a proof assistant. This is unfortunate, because the automated approaches are still very limited in their scalability; the verification of realistic systems appears far out of reach.

A fragment that is often supported by deductive approaches is the special case of k-hypersafety [2, 24, 25, 37], where all executions are quantified universally. This case is attractive, because the verification of a k-hypersafety property can be reduced to the verification of a trace property of the k-fold selfcomposition of the given system.

The general case is much more difficult. Hyperliveness properties like noninference contain an alternation between universal and existential quantification. The $\forall\exists$ pattern occurs in many hyperproperties where the existential quantifier resolves the potential nondeterminism in the system. To prove such hyperproperties, we must, for every execution, establish the existence of another execution such that the two executions respect a specified temporal relation. A straightforward idea is to provide an explicit witness for the existential quantifier (cf. [31]). One way to do this is to view the interaction between the universal and existential quantifiers as an infinite game, such that a winning strategy for the existential player provides a witness for the existential quantifier. For finite-state systems, the winning strategy can be found using techniques from reactive synthesis [18]. In the deductive setting, guessing a winning strategy upfront is not only difficult; any error in the initial choice of strategy might furthermore only be revealed very late in the proof, requiring a restart of the proof from the very beginning. A practical proof technique should therefore be *incremental*, refining the instantiation of the existential quantifier on-the-fly as needed during the proof.

In this paper, we present such an incremental proof technique. We accomplish this by linking temporal hyperproperties to coinductive relations. Coinductive relations are supported by parameterized coinduction [29], a powerful proof technique with built-in support for incremental proofs. This coalgebraic view precisely coincides with the game-theoretic interpretation in the sense that they can prove exactly the same properties. However, the coalgebraic approach is much better suited for an interactive proof assistant. In summary, we make the following contributions:

(1) We introduce HyCo, a coinductive relation to reason about the difficult class of hyperproperties of the form $\forall^*\exists^*\psi$, where ψ is a safety relation between traces. Importantly, HyCo is language agnostic: it does not commit to a specific programming language to model systems, nor to a particular logic to specify trace relations.

(2) We formally prove the soundness of HyCo in the Coq proof assistant. Our proof relies on a subtle use of the axiom of functional choice to resolve the underlying existential quantification.

(3) We further equip HyCo with a collection of incremental reasoning rules, useful up-to techniques, and specialized rules for the case where properties are specified using LTL-style temporal modalities. The soundness of the rules for temporal modalities critically relies on the notion of *derivatives of trace relations*, combined with the idea to reason *up to stronger trace relations*.

(4) Finally, we show that HyCo applies naturally to the verification of reactive systems modeled as imperative programs with I/O. We provide specialized reasoning rules for this case, and give examples of program proofs using HyCo in Coq.

2 Preliminaries

2.1 Infinite Sequences and Set Notations

Given an alphabet Σ, we note Σ^ω the set of all infinite sequences of letters in Σ. Given an infinite sequence $\tau \in \Sigma^\omega$, we note $\tau[i]$ ($i \in \mathbb{N}$) the letter at position i in τ, and we note $\tau[i...]$ the infinite trace obtained by ignoring the first i letters of τ. In particular, if $\tau = \tau_0\tau_1\tau_2...$ we have $\tau[0] = \tau_0$ and $\tau[1...] = \tau_1\tau_2....$ Given two sets A and B, note $A \equiv B$ if A and B are mutually included.

2.2 Labeled Transition Systems

Throughout this paper, we model reactive systems as labeled transition systems (LTS). Formally, a LTS is a triple $(\mathcal{S}, \mathcal{E}, \mathcal{I}, \rightarrow)$ where $\mathcal{S}$ is an arbitrary set of states, $\mathcal{E}$ is an arbitrary set of *observable events*, $\mathcal{I} \in \mathcal{S}$ is an *initial* state, and $\rightarrow \subseteq \mathcal{S} \times \mathcal{E} \cup \{\emptyset\} \times \mathcal{S}$ is a labeled *transition relation* between two states and an (optional) event. Given two states $s_1, s_2 \in \mathcal{S}$, the relation $s_1 \xrightarrow{\emptyset} s_2$ indicates that s_1 can transition to s_2 without emitting any event. When $e \in \mathcal{E}$, $s_1 \xrightarrow{e} s_2$ indicates that s_1 can transition to s_2 while emitting the observable event e. We also adopt the following notation conventions:

$$\begin{array}{cl} \rightarrow & \text{stands for } \xrightarrow{\emptyset} \\ \rightarrow^* & \text{is the reflexive transitive closure of } \rightarrow \\ \overset{e}{\rightsquigarrow} & \text{is a shorthand for } \rightarrow^* \xrightarrow{e} \end{array}$$

Trace Semantics. Given a LTS $TS = (\mathcal{S}, \mathcal{E}, \mathcal{I}, \rightarrow)$, a trace originating from a state $s \in \mathcal{S}$ is an infinite sequence of events $\tau \in \mathcal{E}^\omega$ such that there exists an infinite sequence of states $\pi \in \mathcal{S}^\omega$ with

$$\pi_0 = s \land \forall i, \pi_i \overset{\tau_i}{\rightsquigarrow} \pi_{i+1}$$

Given a state $s \subseteq \mathcal{S}$, we note $\mathit{Traces}_{TS}(s) \subseteq \mathcal{E}^\omega$ the set of traces originating from s. When the transition system being considered is clear from context, we simply note $\mathit{Traces}(s)$ for the traces of s. If $S \subseteq \mathcal{S}$ we note $\mathit{Traces}_{TS}(S) \triangleq \bigcup_{s \in S} \mathit{Traces}(s)$. If $\mathcal{I}$ is the initial state of TS, we note $\mathit{Traces}(TS) \triangleq \mathit{Traces}(\mathcal{I})$ the traces of TS.

2.3 Temporal Hyperproperties

To specify hyperproperties of LTS, we introduce Hyper$_E$, an event-based variant of HyperLTL with support for arbitrary temporal relations between traces. A Hyper$_E$ formula Ψ starts with a sequence of n quantifiers ranging over the traces of some LTS and then specifies a m-ary ($m \geq n$) trace relation ψ.

$$\Psi ::= \forall^{TS}\, \Psi \;\mid\; \exists^{TS}\, \Psi \;\mid\; \psi$$

Given an (ordered) n-tuple of traces Γ, we note $\Gamma \models \Psi$ when Γ satisfies the relation specified by Ψ. The model relation is defined as follows:

$$\begin{aligned} \Gamma \models \forall^{TS}\, \psi &\iff \forall \tau \in \mathit{Traces}(TS), (\Gamma, \tau) \models \psi \\ \Gamma \models \exists^{TS}\, \psi &\iff \exists \tau \in \mathit{Traces}(TS), (\Gamma, \tau) \models \psi \\ \Gamma \models \psi &\iff \Gamma \in \psi \end{aligned}$$

We will often consider relations ψ expressed in (a fragment) of linear temporal logic. The basic assertions are n-ary relations φ between events, and formulas are obtained by using boolean connectives, and the temporal modalities $\mathtt{W}$ (weak-until), $\square$ (always), and $\bigcirc$ (next).

$$\psi ::= \varphi \mid \psi \land \psi \mid \psi \lor \psi \mid \psi \,\mathtt{W}\, \psi \mid \square\psi \mid \bigcirc\psi$$

Given an n-tuple of traces $\Gamma = (\tau_1, ..., \tau_n)$, we note $\Gamma \models \psi$ when Γ satisfies the temporal relation defined by ψ. We note $\Gamma[i]$ the n-tuple of events $(\tau_1[i], ..., \tau_n[i])$, and $\Gamma[i...]$ the n-tuple of traces $(\tau_1[i...], ..., \tau_n[i...])$. The interpretation of the modalities is defined as follows:

$$\begin{aligned}
\Gamma \models \varphi &\iff \Gamma[0] \in \varphi \\
\Gamma \models \bigcirc\psi &\iff \Gamma[1...] \models \psi \\
\Gamma \models \Box\psi &\iff \forall i, \Gamma[i...] \models \psi \\
\Gamma \models \psi_1 \mathtt{W}\, \psi_2 &\iff (\forall i, \Gamma[i...] \models \psi_1) \lor (\exists i, \Gamma[i...] \models \psi_2 \land \forall j < i, \Gamma[j...] \models \psi_1)
\end{aligned}$$

When ψ is defined using combinations of LTL modalities, we freely interchange ψ and its set of models $\{ \Gamma \mid \Gamma \models \psi \}$.

2.4 Parameterized Coinduction

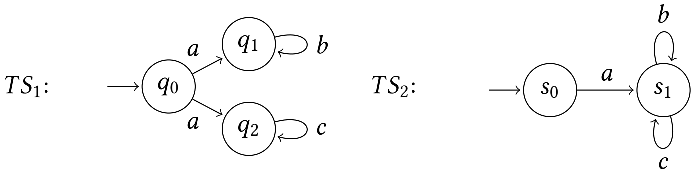
We recall some basics about coinduction. Let $(\mathcal{P}(A), \subseteq, \cup)$ be the complete lattice of subsets of A, and let $F : \mathcal{P}(A) \xrightarrow{mon} \mathcal{P}(A)$ be a monotone operator on subsets. Then, F has a greatest fixed point $\nu.F$ and

$$\nu.F = \bigcup \{ R \mid R \subseteq F(R) \}$$

An immediate consequence of this observation is that whenever we have a property P defined as a greatest fixed point $P = \nu.F_P$ (i.e., defined *coinductively*), we have a systematic method to show that an element x satisfies P: it suffices to prove that x is in some postfixed point R of F_P. In this context, R is usually called the *coinduction hypothesis*.

Lemma 2.1 (Coinduction principle). $X \subseteq \nu.F \iff \exists R, X \subseteq R \land R \subseteq F(R)$

A common use case for coinduction is proofs by simulation. As an example, let us consider the following two finite LTSs:



Clearly, all traces of TS_1 are also valid traces of TS_2 (i.e., TS_1 *refines* TS_2). To prove this fact, one way is to enumerate the traces of TS_1, but this would be impossible if the number of traces was infinite. Another way it to reduce the reasoning about traces to reasoning about states by establishing a *simulation relation*. The idea is to show the existence of a mapping $R \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ between states of TS_1 and TS_2 such that $(q_0, s_0) \in R$, and such that for every pair of states $(q, s) \in R$, if $q \overset{e}{\rightsquigarrow} q'$ then there exists a state s' with $s \overset{e}{\rightsquigarrow} s'$ and (q', s') is again in R. The existence of such a mapping ensures that for all traces originating from q_0, there is a way to reproduce the same trace from s_0. This intuition can be understood in coalgebraic terms by observing that the existence of a simulation relation R containing (q_0, s_0) simply amounts to proving $(q_0, s_0) \in \mathtt{sim}$ where $\mathtt{sim}$ is the coinductive relation defined as follows:

$$\begin{aligned}
\mathtt{simF}(R) &\triangleq \{(q, s) \mid \forall e. \forall q'. q \overset{e}{\rightsquigarrow} q' \implies \exists s', s \overset{e}{\rightsquigarrow} s' \land (q', s') \in R\} \\
\mathtt{sim} &\triangleq \nu.\mathtt{simF}
\end{aligned}$$

By Lemma 2.1, $(q_0, s_0) \in \mathtt{sim}$ if and only if there exists some relation R containing (q_0, s_0) and such that $R \subseteq \mathtt{simF}(R)$. Going back to our simple example, $R = \{(q_0, s_0), (q_1, s_1), (q_2, s_1)\}$ is a postfixed point of $\mathtt{simF}$, which proves trace-inclusion.

For our previous example, guessing a simulation R was not too difficult. For larger examples, especially if we consider transition systems with infinite state-spaces, finding R can be extremely tedious. Instead we could also start with $R = \{(q_0, s_0)\}$, and explore the transition systems from there, carefully matching transitions in the left-hand transition system with corresponding transitions in the right-hand transition system and adding pairs of states to R until we reach a postfixed point. To achieve this incremental proof style, Hur et al. proposed *parameterized coinduction* [29]. The key idea is to replace the greatest fixed point $\nu.F$ with a parameterized variant $G_F(H)$ where H is a current *guess* for the coinduction hypothesis R. The *parameterized greatest fixed point* operator is defined as

$$G_F(H) \triangleq \nu.(\lambda X.F(X \cup H))$$

By definition, we have $\nu.F \equiv G_F(\emptyset)$, which means that any time we need to prove a theorem of the form $X \subseteq \nu.F$, we can prove $X \subseteq G_F(\emptyset)$ instead. An important difference, however, is that G_F can be equipped with the following set of *incremental* reasoning rules:

$$\begin{array}{ccc}
\textsc{Init} & \textsc{Accumulate} & \textsc{Step} \\
\dfrac{X \subseteq G_F(\emptyset)}{X \subseteq \nu F} & \dfrac{X \subseteq G_F(H \cup X)}{X \subseteq G_F(H)} & \dfrac{X \subseteq F(H \cup G_F(H))}{X \subseteq G_F(H)}
\end{array}$$

Fig. 1. Rules of parameterized coinduction

Briefly,

(1) The rule Init exploits the equation $\nu F \equiv G_F(\emptyset)$ to initiate a proof by parameterized coinduction.
(2) The rule Accumulate allows us to incrementally extend the current guess H
(3) The rule Step allows us to make progress in the proof by unfolding the fixed point equation $G_F(H) = F(H \cup G_F(H))$

We refer the reader to [29] for a more detailed explanation of parameterized coinduction and for proofs of these reasoning rules. For now, let us go back to our simulation example, and give an incremental proof by parameterized coinduction.

$$\begin{aligned}
& (q_0, s_0) \in \nu.\mathtt{simF} \\
\Longleftrightarrow\; & (q_0, s_0) \in G_{\mathtt{simF}}(\emptyset) && \text{(by Init)} \\
\Longleftarrow\; & \underbrace{\exists s', s_0 \overset{a}{\rightsquigarrow} s' \wedge (q_1, s') \in G_{\mathtt{simF}}(\emptyset)}_{\text{find a step matching } q_0 \xrightarrow{a} q_1} \wedge \underbrace{\exists s', s_0 \overset{a}{\rightsquigarrow} s' \wedge (q_2, s') \in G_{\mathtt{simF}}(\emptyset)}_{\text{find a step matching } q_0 \xrightarrow{a} q_2} && \text{(by Step)} \\
\Longleftarrow\; & (q_1, s_1) \in G_{\mathtt{simF}}(\emptyset) \wedge (q_2, s_1) \in G_{\mathtt{simF}}(\emptyset) && (\text{pick } s' = s_1) \\
\Longleftarrow\; & (q_1, s_1) \in G_{\mathtt{simF}}(\{(q_1, s_1)\}) \wedge (q_2, s_1) \in G_{\mathtt{simF}}(\{(q_2, s_1)\}) && \text{(by Accumulate)} \\
\Longleftarrow\; & \underbrace{\exists s', s_1 \overset{b}{\rightsquigarrow} s' \wedge (q_1, s') \in \{(q_1, s_1)\} \cup G_{\mathtt{simF}}(\{(q_1, s_1)\})}_{\text{find a step matching } q_1 \xrightarrow{b} q_1} \wedge && \text{(by Step)} \\
& \underbrace{\exists s', s_1 \overset{c}{\rightsquigarrow} s' \wedge (q_2, s') \in \{(q_2, s_1)\} \cup G_{\mathtt{simF}}(\{(q_2, s_1)\})}_{\text{find a step matching } q_2 \xrightarrow{c} q_2} \\
\Longleftarrow\; & (q_1, s_1) \in \{(q_1, s_1)\} \wedge (q_2, s_1) \in \{(q_2, s_1)\} && (\text{pick } s' = s_1)
\end{aligned}$$

3 Hyperliveness and Relational Invariance, Coinductively

In the previous section, we recalled that trace inclusion between two transition systems can be reduced to reasoning about states by finding a simulation relation. Further, parameterized coinduction can be used to construct a simulation relation incrementally. Trace inclusion is an example of hyperliveness property: given two transition systems TS_1 and TS_2 emitting events in $\mathcal{E}$, it can be specified as the formula $\forall^{TS_1}\exists^{TS_2}\Box\, eq$ where $eq = \{\ (e, e) \mid e \in \mathcal{E}\ \}$ is equality of events. In this section, we show that parameterized coinduction can also be used for incremental proofs of any hyperproperties of the form $\forall\exists\Box\varphi$ where φ is a binary relation between events. To do so, we generalize the operator $\mathtt{simF}$ and define a coinductive relation $\mathtt{fei}_\varphi$ that soundly underapproximates $\forall\exists\Box\varphi$. We describe the construction of $\mathtt{fei}_\varphi$, show that it can be equipped with useful reasoning rules, and propose a formal proof of soundness via the axiom of functional choice. In Section 4, we extend this construction to support the more general class of $\forall\exists\psi$ hyperproperties where ψ is a safety relation between traces.

3.1 Generalizing Simulation Relations

For the remainder of this section, let $TS_1 = (\mathcal{S}_1, \mathcal{E}_1, \mathcal{I}_1, \rightarrow_1)$ and $TS_2 = (\mathcal{S}_2, \mathcal{E}_2, \rightarrow_2)$ be two LTSs, and let us fix a binary relation between events $\varphi \subseteq \mathcal{E}_1 \times \mathcal{E}_2$. We define the relation $\mathtt{fei}_\varphi$ (**f**orall, **e**xists, **i**nvariant) as follows:

$$\mathtt{feiF}_\varphi(R) \triangleq \{\ (s_1, s_2) \mid \forall s_1 \overset{e_1}{\rightsquigarrow} s_1' \implies \exists s_2 \overset{e_2}{\rightsquigarrow} s_2' \wedge (e_1, e_2) \in \varphi \wedge R(s_1', s_2')\ \}$$
$$\mathtt{fei}_\varphi \triangleq \nu.\mathtt{feiF}_\varphi$$

As expected, the $\mathtt{fei}_\varphi$ relation provides a sound proof technique to verify hyperproperties of the form $\forall\exists\Box\varphi$. More precisely, if the initial states of TS_1 and TS_2 are related by $\mathtt{fei}_\varphi$, it can then be concluded that $\forall^{TS_1}\exists^{TS_2}\Box\varphi$ is valid.

Theorem 3.1 (Soundness of $\mathtt{fei}_\varphi$). *Let TS_1 and TS_2 be two LTSs and let $\varphi \subseteq \mathcal{E}_1 \times \mathcal{E}_2$. We have*

$$(\mathcal{I}_1, \mathcal{I}_2) \in \mathtt{fei}_\varphi \implies \models \forall^{TS_1}\exists^{TS_2}\Box\varphi$$

While this theorem might seem like an unsurprising generalization of simulation proofs, its formal proof turns out to be much more involved. Indeed, to prove this theorem we need to find a suitable trace to instantiate the inner existential quantifier. Note that in the simpler case of simulation proofs, since the two quantified traces are required to be equal, the existential quantifier is superfluous and it suffices to instantiate it with a copy of the universal trace. In the more general case of $\forall\exists\Box\varphi$, the witness for the existential quantifier depends on the relation φ (and the quantified systems) and we need to somehow *extract* it from the proof of $(\mathcal{I}_1, \mathcal{I}_2) \in \mathtt{fei}_\varphi$. We delay the explanation of this process to Section 3.4.

3.2 Incremental Proofs

As discussed in the preliminaries, a limitation of standard coinductive proofs is that they require us to find a coinduction hypothesis up-front. Instead, parameterized coinduction [29] allows us to perform coinductive proofs *incrementally* by starting from $\emptyset$ and progressively guessing portions of a coinductive hypothesis until a postfixed point is reached. In this section, we leverage parameterized coinduction to prove hyperproperties incrementally via $\mathtt{fei}$. Taking inspiration from notations introduced in [15], we start by defining the following two *semantic quadruples*:

$$\boxed{H} \vdash_{\mathtt{inv}} s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \varphi\ \} \triangleq (s_1, s_2) \in G_{\mathtt{feiF}_\varphi}(H)$$
$$\dbox{H} \vdash_{\mathtt{inv}} s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \varphi\ \} \triangleq (s_1, s_2) \in H \cup G_{\mathtt{feiF}_\varphi}(H)$$

Note that $(s_1, s_2) \in \mathtt{fei}_\varphi \iff \boxed{\emptyset} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \; \{ \; \varphi \; \}$. In turn, to prove $\models \forall^{TS_1} \exists^{TS_2} \square \varphi$, it suffices to show $\boxed{\emptyset} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \; \{ \; \varphi \; \}$. Further, since these quadruples are defined in terms of the parameterized greatest fixed point operator G, they admit incremental reasoning principles. The intention behind the choice of notation is that $\boxed{H}$ represents a *guarded* coinduction hypothesis (i.e., it cannot yet be used to conclude a proof), whereas $\dbox{H}$ represents an *unguarded* coinduction hypothesis (i.e., it can be used to immediately conclude a proof if $(s_1, s_2) \in H$). The core reasoning rules associated with these quadruples are presented in Figure 2. In the conclusion of some rules, we note $H^?$ instead of $\dbox{H}$ or $\boxed{H}$ when the rule can be applied regardless of whether the hypothesis is currently guarded or not.

$$\textsc{Init}\quad \frac{\boxed{\emptyset} \vdash_{\mathtt{inv}} {\mathcal{I}_1}^{\forall} \mid {\mathcal{I}_2}^{\exists} \; \{ \; \varphi \; \}}{\models \forall^{TS_1} \exists^{TS_2} \square \varphi} \qquad \textsc{Cycle}\quad \frac{(s_1, s_2) \in H}{\dbox{H} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \; \{ \; \varphi \; \}}$$

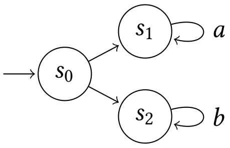
$$\textsc{Step}\quad \frac{\forall s_1 \overset{e_1}{\rightsquigarrow} s_1',\ \exists s_2 \overset{e_2}{\rightsquigarrow} s_2',\ (e_1, e_2) \in \varphi \wedge \dbox{H} \vdash_{\mathtt{inv}} {s_1'}^{\forall} \mid {s_2'}^{\exists} \; \{ \; \varphi \; \}}{H^? \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \; \{ \; \varphi \; \}}$$

$$\textsc{Invariant}\quad \frac{(s_1, s_2) \in H' \qquad \forall (s_1, s_2) \in H',\ \boxed{H \cup H'} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \; \{ \; \varphi \; \}}{H^? \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \; \{ \; \varphi \; \}}$$

Fig. 2. Incremental reasoning rules for fei

All rules are derived from the soundness of fei (Theorem 3.1), the rules of parameterized coinduction (Figure 1), and the definition of feiF. The rule Init exploits the soundness of fei to initiate a proof by parameterized coinduction. When focusing on a pair of states (s_1, s_2), the rule Step requires to prove that for every transition $s_1 \overset{e_1}{\rightsquigarrow} s_1'$, there exists a corresponding transition $s_2 \overset{e_2}{\rightsquigarrow} s_2'$ such that e_1 and e_2 satisfy the event-invariant φ. Applying Step transfers the focus to the pairs of states (s_1', s_2') and releases the guard. Cycle allows us to finish a proof by creating a cycle: if we already encountered a pair (s_1, s_2) earlier in the proof (i.e., if $(s_1, s_2) \in H$) we can use this fact to immediately conclude. Note that applying Cycle requires the hypothesis to be unguarded. Finally, the rule Invariant is a reformulation of the rule Accumulate of parameterized coinduction. It extends the current coinduction hypothesis H with a larger guess $H \cup H'$, where H' contains at least the current pair of states. The cost to pay is that applying Invariant restores the guard, and requires to prove a quadruple for every state in $H \cup H'$ instead of just the initial pair of states. The benefit is that, after applying the rule Invariant, we can conclude a proof as soon as we reach again a pair of states that satisfy the *invariant* H'. We note that the rules Step and Invariant can be applied regardless of whether the hypothesis is currently guarded or not.

Example 3.2. Let us consider the following transition system, whose events are $\mathcal{E} = \{a, b\}$:



Suppose we wish to prove $\models \forall^{TS}\exists^{TS}\square(a_1 \leftrightarrow b_2)$ where $a_1 \leftrightarrow b_2$ stands for the binary relation $\{\ (a,b),(b,a)\ \}$. We give an incremental proof using the rules of Figure 2. The proof starts by exploring the system until we reach states s_1 and s_2:

$$\begin{array}{rlr} & \models \forall^{TS}\exists^{TS}\square(a_1 \leftrightarrow b_2) & \\ \Longleftarrow & \boxed{\emptyset} \vdash_{\mathtt{inv}} {s_0}^{\forall} \mid {s_0}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} & \text{(by Init)} \\ \Longleftarrow & \dbox{\emptyset} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} \wedge \dbox{\emptyset} \vdash_{\mathtt{inv}} {s_2}^{\forall} \mid {s_1}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} & \text{(by Step)} \end{array}$$

Then, we observe that the only way out of the pair (s_1, s_2) (resp. (s_2, s_1)) is to go back to (s_1, s_2) (resp. (s_2, s_1)). This indicates that $H_1 = \{(s_1, s_2)\}$ and $H_2 = \{(s_2, s_1)\}$ are good choices of coinduction hypotheses. We therefore extend our current hypotheses with H_1 and H_2 using Invariant:

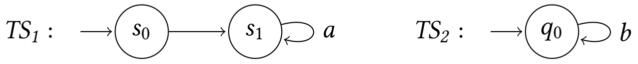
$$\begin{array}{rlr} & \dbox{\emptyset} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} \wedge \dbox{\emptyset} \vdash_{\mathtt{inv}} {s_2}^{\forall} \mid {s_1}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} & \\ \Longleftarrow & \boxed{H_1} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} \wedge \boxed{H_2} \vdash_{\mathtt{inv}} {s_2}^{\forall} \mid {s_1}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} & \text{(by Invariant)} \end{array}$$

Using Step, a-transitions out of s_1 are matched with b-transitions out of s_2 (and vice-versa), thus maintaining the event-invariant $a_1 \leftrightarrow b_2$. Note that taking a step releases the guard! Since transitioning out of (s_1, s_2) (resp. (s_2, s_1)) cycles back to (s_1, s_2) (resp. (s_2, s_1)), we can then conclude with Cycle:

$$\begin{array}{rlr} & \boxed{H_1} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} \wedge \boxed{H_2} \vdash_{\mathtt{inv}} {s_2}^{\forall} \mid {s_1}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} & \\ \Longleftarrow & \dbox{H_1} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} \wedge \dbox{H_2} \vdash_{\mathtt{inv}} {s_2}^{\forall} \mid {s_1}^{\exists} \ \{\ a_1 \leftrightarrow b_2\ \} & \text{(by Step)} \\ \Longleftarrow & (s_1, s_2) \in H_1 = \{(s_1, s_2)\} \wedge (s_2, s_1) \in H_2 = \{(s_2, s_1)\} & \text{(by Cycle)} \end{array}$$

3.3 Alignment Rules

Sometimes, it is convenient to be able to step through the execution from the left state and the right state at different paces in order to align the states in a certain way before establishing an invariant via Invariant. As an example, consider the following two transition systems:



Clearly, $\models \forall^{TS_1}\exists^{TS_2}\square(a_1 \wedge b_2)$. One way to prove it is to use the Step rule to reach (s_1, q_0), then Invariant to add (s_1, q_0) to the current coinduction hypothesis, and conclude by using Step and Cycle. While this proof is perfectly valid, it forces us to use the Step rule two times to prove that an a on the left-hand transition system can always be matched by a b in the right-hand system. For such a small example, duplicating the reasoning is not difficult, but for more complicated systems, it might actually be extremely tedious. Instead, what we would like to do is to first align the states s_1 and q_0 by taking one step of computation in the left-hand transition system, then add (s_1, q_0) to the current hypothesis, and finish the proof with a single application of the rule Step followed by Cycle. In this section, we introduce reasoning rules to support this intuition.

To reproduce the proof we just described, we would need a rule of the following form:

$$\frac{s_1 \rightarrow^* s_1' \qquad \boxed{H} \vdash_{\mathtt{inv}} {s_1'}^{\forall} \mid {s_2}^{\exists} \ \{\ \varphi\ \}}{\boxed{H} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \ \{\ \varphi\ \}}$$

This rule would allow us to skip silent computation steps in the left-hand transition system. Unfortunately, it is unsound in the presence of nondeterminism. Indeed, if along the path from s_1 to s_1', there are possibilities to branch out to some other state s_1'', this rule would not ensure that the step to s_1'' can be matched by a corresponding step from s_2. A potential solution would be to *determinize* the rule by universally quantifying on the post-states of s_1:

$$\frac{\forall s_1', s_1 \to^* s_1' \implies \boxed{H} \vdash_{\mathtt{inv}} {s_1'}^{\forall} \mid {s_2}^{\exists} \; \{\, \varphi \,\}}{\boxed{H} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\, \varphi \,\}}$$

This rule is naturally sound, as it forces us to cover all possible ways to transition out of s_1. However, instead of solving the initial problem, it makes it even worse by generating redundant proof obligations. Indeed, if we have a path $s_1 \to s_1^1 \to s_1^2 ... \to s_1'$, this rule will generate a proof obligation for all s_1^i. Instead, what we would like is to merge as many steps as possible to minimize the number of proof obligations. To achieve this goal, we need to be able to distinguish *deterministic* sequences of computations from *nondeterministic* ones. To this effect, we introduce the *deterministic* transition relation $\to_{det}$ defined as follows:

$$s \to_{det} s' \iff s \to s' \wedge (\forall e \forall s'', s \xrightarrow{e} s'' \implies s' = s'' \wedge e = \emptyset)$$

Intuitively, a deterministic transition $s_1 \to_{det} s_2$ indicates that s_2 is the only possible successor of s_1 and further, that no event can be emitted by the transition to s_2. Using $\to_{det}$ instead of $\to$, we obtain the following sound rules:

$$\textsc{Steps-L}\quad \frac{s_1 \to_{det}^* s_1' \qquad \boxed{H} \vdash_{\mathtt{inv}} {s_1'}^{\forall} \mid {s_2}^{\exists} \; \{\, \varphi \,\}}{\boxed{H} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\, \varphi \,\}} \qquad \textsc{Steps-R}\quad \frac{s_2 \to^* s_2' \qquad \boxed{H} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2'}^{\exists} \; \{\, \varphi \,\}}{\boxed{H} \vdash_{\mathtt{inv}} {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\, \varphi \,\}}$$

3.4 Soundness Proof

The initial claim that motivated the previous section is that $\mathtt{fei}_\varphi$ is a sound approximation of the hyperproperty $\forall\exists\Box\varphi$. This section is dedicated to the proof of this fact. While the high-level intuition is relatively straightforward, the formal proof turns out to be more involved. We start by giving an intuitive informal proof, and we then present a formal proof via the axiom of functional choice. The latter has been mechanized in the Coq proof assistant.

Informal proof. Let $\varphi \subseteq \mathcal{E}_1 \times \mathcal{E}_2$ be a relation on events, $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$ be two states, and assume that $(s_1, s_2) \in \mathtt{fei}_\varphi$. Under these assumptions, we would like to prove

$$\forall \tau_1 \in \mathit{Traces}(s_1), \exists \tau_2 \in \mathit{Traces}(s_2), (\tau_1, \tau_2) \in \Box\varphi$$

Given a trace $\tau_1 \in \mathit{Traces}(s_1)$, we need to construct an appropriate trace τ_2 to instantiate the existential quantifier. Since $\tau_1 \in \mathit{Traces}(s_1)$, there has to exist some state s_1' such that

$$(1)\; s_1 \overset{\tau_1[0]}{\rightsquigarrow} s_1' \qquad (2)\; \tau_1[1...] \in \mathit{Traces}(s_1')$$

Since $(s_1, s_2) \in \mathtt{fei}_\varphi = \nu.\mathtt{feiF}_\varphi$, by unfolding the fixed point equation $\mathtt{fei}_\varphi = \mathtt{feiF}_\varphi(\nu.\mathtt{feiF}_\varphi)$, unfolding the definition of $\mathtt{feiF}$, and using (1) we obtain the existence of an event e_2^0 and a state s_2' such that

$$(3)\; s_2 \overset{e_2^0}{\rightsquigarrow} s_2' \qquad (4)\; (\tau_1[0], e_2^0) \in \varphi \qquad (5)\; (s_1', s_2') \in \mathtt{fei}_\varphi$$

We can repeat this reasoning from the assumption that $(s_1', s_2') \in \mathtt{fei}_\varphi$ and from (2) to obtain a state s_1'', and event e_2^1, and a state s_2'' such that

$$(6)\; s_2' \overset{e_2^1}{\rightsquigarrow} s_2'' \qquad (7)\; (\tau_1[1], e_2^1) \in \varphi \qquad (8)\; (s_1'', s_2'') \in \mathtt{fei}_\varphi$$

Iterating this process an infinite number of times, we obtain a sequence of events $\tau_2 = e_2^0 e_2^1 e_2^2 ....$ By construction $\tau_2 \in \mathit{Traces}(s_2)$ and clearly $(\tau_1, \tau_2) \in \Box\varphi$ since $(\tau[i], e_2^i) \in \varphi$.

Formal Proof via Classical Choice. The main challenge when mechanizing the previous proof in the Coq proof assistant is to formalize the intuition of "iterating this process an infinite number of times". We propose a solution in four stages:

(1) We start by defining an abstract labeled transition system between *proof states*. Transitions are labeled by events.
(2) We prove that this abstract system is progressive: for any proof state, we can always transition to a successor state.
(3) Using the axiom of functional choice and (2), we extract a deterministic and executable version of the abstract transition system that produces exactly one trace
(4) Using (3), we compute the unique trace of the abstract transition system. The system is constructed in such a way that this trace is guaranteed to be a valid witness for the existential quantifier.

Given a relation $\varphi \subseteq \mathcal{E}_1 \times \mathcal{E}_2$, we define the set of *proof states* Π_φ as follows:

$$\Pi_\varphi \subseteq \mathcal{S}_1 \times \mathcal{S}_2 \times \mathcal{S}_1^\omega \triangleq \{\ (s_1, s_2, \tau_1) \mid \tau_1 \in \mathit{Traces}(s_1) \land (s_1, s_2) \in \mathtt{fei}_\varphi\ \}$$

We then define the following labeled *proof-step* relation $\overset{-}{\longrightarrow}_\varphi \subseteq \Pi_\varphi \times \mathcal{E}_2 \times \Pi_\varphi$:

$$\frac{\tau_1 = e_1 \cdot \tau_1' \qquad s_1 \overset{e_1}{\rightsquigarrow} s_1' \qquad s_2 \overset{e_2}{\rightsquigarrow} s_2' \qquad (e_1, e_2) \in \varphi}{(s_1, s_2, \tau_1) \xrightarrow{e_2}_\varphi (s_1', s_2', \tau_1')}$$

This transition relation describes an abstract transition system over proof states, and such that each transition generates an event in $\mathcal{E}_2$. Importantly, it can be proven that this abstract transition system is progressive: every proof state has a successor.

Lemma 3.3 (Progress). $\forall \pi \in \Pi_\varphi, \exists (e_2, \pi') \in \mathcal{E}_2 \times \Pi_\varphi, \pi \xrightarrow{e_2}_\varphi \pi'$

Proof. Let $(s_1, s_2, \tau_1) \in \Pi_\varphi$. By definition we have $\tau_1 \in \mathit{Traces}(s_1)$ and $(s_1, s_2) \in \mathtt{fei}_\varphi$. By unfolding the the definition of *Traces*, we easily obtain the existence of a state s_1' such that

$$s_1 \overset{\tau_1[0]}{\rightsquigarrow} s_1' \land \tau_1[1...] \in \mathit{Traces}(s_1')$$

By unfolding the definition of $\mathtt{fei}_\varphi$ and using the fact that $s_1 \overset{\tau_1[0]}{\rightsquigarrow} s_1'$, we obtain the existence of an event e_2 and a state s_2' such that

$$s_2 \overset{e_2}{\rightsquigarrow} s_2' \land (\tau_1[0], e_2) \in \varphi \land (s_1', s_2') \in \mathtt{fei}_\varphi$$

Clearly, we have $(s_1', s_2', \tau[1...]) \in \Pi_\varphi$, and $(s_1, s_2, \tau_1) \xrightarrow{e_2}_\varphi (s_1', s_2', \tau_1[1...])$, which concludes our proof. □

By the axiom of functional choice, the previous lemma guarantees the existence of a function $\mathtt{progress} : \Pi_\varphi \to \mathcal{E}_2 \times \Pi_\varphi$ that computes an event and a successor state. We start by recalling the axiom in set-theoretic terms for clarity (the Coq implementation of the proof relies on a type-theoretic formulation of the same axiom).

Definition 3.4 (Functional Choice). Let A and B be two sets and $R \subseteq A \times B$ a binary relation such that $\forall a \in A, \exists b \in B, (a, b) \in R$. Then, there exists a function $f : A \to B$ such that $\forall a \in A, (a, f(a)) \in R$

Corollary 3.5. *There exists a function* $\mathtt{progress}_\varphi : \Pi_\varphi \to \mathcal{E}_2 \times \Pi_\varphi$ *such that for any proof state* π *we have*

$$(e, \pi') = \mathtt{progress}_\varphi(\pi) \implies \pi \xrightarrow{e}_\varphi \pi'$$

Using the progress function provided by this corollary, for any proof state π we can construct a sequence $\texttt{witness}(\pi) \in \mathcal{E}_2$ defined coinductively as follows:

$$\texttt{witness}(\pi) \triangleq \textbf{let } (e, \pi') = \texttt{progress}(\pi) \textbf{ in } e \cdot \texttt{witness}(\pi')$$

By construction, the witness function satisfies the following properties.

Lemma 3.6 (Witness). *Let $\pi = (s_1, s_2, \tau_1) \in \Pi_\varphi$ and let $\tau_2 = \texttt{witness}(\pi)$. Then $\tau_2 \in Traces(s_2)$ and $(\tau_1, \tau_2) \in \square\varphi$*

Proof. Both facts follow from the definition of the proof-step relation $\longrightarrow_\varphi$. Indeed, if we have a sequence of proof states

$$(s_1^0, s_2^0, \tau_1^0) \xrightarrow{e_2^0}_\varphi (s_1^1, s_2^1, \tau_1^1) \xrightarrow{e_2^1}_\varphi (s_1^2, s_2^2, \tau_1^2) \ \dots$$

then in particular $\forall i \in \mathbb{N}, s_2^i \overset{e_2^i}{\rightsquigarrow} s_2^{i+1}$ (thus proving that τ_2 is a valid trace), and $\forall i \in \mathbb{N}, (\tau_1^i[0], e_2^i) \in \varphi$ (thus proving that $(\tau_1, \tau_2) \in \square\varphi$). □

We can now easily prove the soundness of $\texttt{fei}_\varphi$ (Theorem 3.1).

Proof. Suppose $(s_1, s_2) \in \texttt{fei}_\varphi$, and let $\tau_1 \in Traces(s_1)$, we show

$$\exists\tau_2 \in Traces(s_2), (\tau_1, \tau_2) \in \square\varphi$$

We pose $\pi \triangleq (s_1, s_2, \tau_1)$. Clearly, $\pi \in \Pi_\varphi$. We then choose $\tau_2 = \texttt{witness}(\pi)$. By the previous lemma, $\tau_2 \in Traces(s_2)$ and $(\tau_1, \tau_2) \in \square\varphi$. If we consider $s_1 = \mathcal{I}_1$ and $s_2 = \mathcal{I}_2$, the above construction gives a proof of $\models \forall^{TS_1}\exists^{TS_2}\square\varphi$. □

4 From Relational Invariants to Safety Relations

While formulas of the form $\forall\exists\square\varphi$ are already covering relevant hyperproperties such as trace-inclusion and several variants of generalized non-interference, many temporal hyperproperties of interest are of the more general form $\forall\exists\psi$ where ψ is an arbitrary *safety relation* between traces. In this section, we extend the coinductive relation fei to support reasoning about arbitrary safety relations. Further, we develop reasoning rules for the specific case where safety relations are defined using LTL-style temporal modalities.

4.1 It's All About Staying Safe

Intuitively, safety relations are relations that specify the absence of bad interactions between two traces. More formally, safety relations between infinite traces are exactly these for which it suffices to look at finite prefixes to determine whether the relation is violated. In other words, for every safety relation there exists a so called *monitor* that, given a pair of trace prefixes, checks wether or not they are compatible with the relation. Further, two traces satisfy a safety relation if and only if all their prefixes are accepted by the monitor. Following this intuition, establishing a safety property ψ can always be reduced to a proof of invariance: it suffices to run the system and the monitor for ψ concurrently, and to prove the invariant "the monitor never reports any violation". However, extracting an appropriate monitor for any arbitrary safety relation ψ is non-trivial, especially when the relation ψ is expressed in an expressive temporal logic. One way to systematically extract a monitor from a high-level logical specification is by computing a finite-word *safety automaton* that accepts all pairs of prefixes compatible with a trace relation. While this approach works well for automated verification (because verification can then be reduced to efficient operations on automata), it it not a satisfactory solution for interactive verification inside a proof assistant. First of all, translating a high-level specification expressed in a temporal logic to an equivalent automaton is a non-trivial task. Further, we would also need to formally verify the correctness of this translation. Finally, even assuming that we have a certified algorithm to translate high-level specifications to automata [23], we would then need to reason about states and transitions of these automata rather than reasoning about the original logical specifications. This would lead to tedious and unintuitive proofs. Instead, we propose a solution via the notion of *property derivatives* [14, 39]. The key insight is to observe that given a safety relation ψ and two traces τ_1, τ_2, there is a systematic way to decompose a proof of $(\tau_1, \tau_2) \in \psi$ into a condition on $\tau_1[0], \tau_2[0]$ expressing *what needs to hold now* in order for ψ to not be immediately violated; and a proof of $(\tau_1[1...], \tau_2[1...]) \in \psi'$ for some carefully chosen ψ' expressing what need to hold *next* in order for ψ to not be violated later. In this section, we exploit this observation to extend the coinductive relation `fei` presented in the previous section to a more general relation `fe` that supports reasoning about arbitrary safety relations.

4.2 Coinductive Safety Proofs

Formally, safety relations between traces can be defined in terms of a *safety closure* operator $|-|_{safe} : \mathcal{P}(\mathcal{E}_1^\omega \times \mathcal{E}_2^\omega) \xrightarrow{mon} \mathcal{P}(\mathcal{E}_1^\omega \times \mathcal{E}_2^\omega)$. Given a trace relation ψ, the safety closure $|\psi|_{safe}$ is extracting the *safety* part of ψ by relating all pairs of traces such that all their prefixes are *compatible* with ψ (i.e., there exists a way to extend the prefixes into infinite traces that satisfy ψ).

Definition 4.1 (Safety closure). Let $\psi \subseteq \mathcal{E}_1^\omega \times \mathcal{E}_2^\omega$ be a binary trace relation. The safety closure of ψ, noted $|\psi|_{safe}$ is the binary trace relation defined as follows:

$$|\psi|_{safe} = \{(\tau_1, \tau_2) \mid \forall n, \exists \tau_1', \tau_2' \in \mathcal{E}_1^\omega \times \mathcal{E}_2^\omega, (\tau_1[n...] \cdot \tau_1', \tau_2[n..] \cdot \tau_2') \in \psi\}$$

Safety properties are then defined to be all fixed points of $|-|_{safe}$, i.e., all the trace relations that are equivalent to their safety closure. Since $\psi \subseteq |\psi|_{safe}$ for any ψ, safety relations can be equivalently characterized as prefixed points of $|-|_{safe}$.

Definition 4.2 (Safety relations). A trace relation ψ is said to be a *safety* relation iff $|\psi|_{safe} \subseteq \psi$

We observe that the safety closure of a relation can equivalently be defined as a greatest fixed point. We start by introducing an operator Δ, inspired by the notion of *derivatives* introduced by Brzozowski [14]. Given two events $e_1 \in \mathcal{E}_1, e_2 \in \mathcal{E}_2$, the derivative of ψ, noted $\Delta_{e_1,e_2}(\psi)$, is the relation that should be satisfied by suffixes $\tau_1 \in \mathcal{E}_1^\omega, \tau_2 \in \mathcal{E}_2^\omega$ in order for the traces $(e_1\tau_1, e_2\tau_2)$ to satisfy ψ. Formally, we have:

$$\Delta_{e_1,e_2}(\psi) \triangleq \{ (\tau_1, \tau_2) \mid (e_1\tau_1, e_2\tau_2) \in \psi \}$$

Two traces $\tau_1 = e_1\tau_1'$ and $\tau_2 = e_2t_2'$ are in the safety closure of φ if the derivative $\Delta_{e_1,e_2}(\psi)$ is non-empty (i.e., the first two symbols are not trivially violating the relation), and if the suffixes τ_1', τ_2' are again in the closure of $\Delta_{e_1,e_2}(\psi)$. This intuition is formally captured by the definition of a ternary coinductive relation $\mathtt{safe} \in \mathcal{P}(\mathcal{E}_1^\omega \times \mathcal{E}_2^\omega \times \mathcal{P}(\mathcal{E}_1^\omega \times \mathcal{E}_2^\omega))$

$$\begin{aligned}\mathtt{safeF}(C) &\triangleq \{(e_1\tau_1, e_2\tau_2, \psi) \mid \Delta_{e_1,e_2}(\varphi) \neq \emptyset \wedge (\tau_1, \tau_2, \Delta_{e_1,e_2}(\psi)) \in C\} \\ \mathtt{safe} &\triangleq \nu.\mathtt{safeF}\end{aligned}$$

The relation `safe` precisely characterizes the safety closure in the sense of the following lemma:

Lemma 4.3. $(\tau_1, \tau_2) \in |\psi|_{safe} \iff (\tau_1, \tau_2, \psi) \in \mathtt{safe}$

Proof.

- $(\Longleftarrow)$ By induction on the length of the prefixes in the definition of $|-|_{safe}$, unfolding the fixed point equation $\mathtt{safe} = \mathtt{safeF}(\mathtt{safe})$, and exploiting the fact that $\Delta_{e_1,e_2}(\psi) \neq \emptyset \iff \exists (\tau_1, \tau_2), (e_1\tau_1, e_2\tau_2) \in \psi$ $(*)$.

- ($\Longrightarrow$) By coinduction with $R = \{ (\tau_1, \tau_2, \psi) \mid (\tau_1, \tau_2) \in \psi \}$ as the coinduction hypothesis. The fact that R is a postfixed point of $\mathtt{safeF}$ follows from $(*)$ and the additional observation that $(e_1\tau_1, e_2\tau_2) \in |\psi|_{safe} \implies (\tau_1, \tau_2) \in |\Delta_{e_1,e_2}(\psi)|_{safe}$.

□

As an immediate corollary, we can use coinduction to prove that a pair of traces satisfies any safety relation.

Corollary 4.4 (Coinductive safety proofs). *Let ψ be a safety property, then*

$$(\tau_1, \tau_2) \in \psi \iff (\tau_1, \tau_2) \in |\psi|_{safe} \iff (\tau_1, \tau_2, \psi) \in \mathtt{safe}$$

A useful interpretation of this corollary is that, given two traces τ_1, τ_2 and a safety property ψ, the derivative operator gives a systematic way to split the proof of $(\tau_1, \tau_2) \in \psi$ into *what needs to hold now* (i.e., $\Delta_{\tau_1[0],\tau_2[0]}(\psi) \neq \emptyset$) and *what needs to hold next* (i.e., $(\tau_1[1...], \tau_2[1...], \Delta_{\tau_1[0],\tau_2[0]}(\psi)) \in \mathtt{safe}$). The next subsection builds on this intuition to construct a coinductive relation $\mathtt{fe}$ that extends $\mathtt{fei}$ with support for reasoning about arbitrary safety relations.

4.3 A Coinductive Relation Supporting Safety Specifications

For the rest of this section, we fix two LTSs $TS_1 = (\mathcal{S}_1, \mathcal{E}_1, \mathcal{I}_1, \rightarrow_1)$ and $TS_2 = (\mathcal{S}_2, \mathcal{E}_2, \mathcal{I}_2, \rightarrow_2)$. To extend $\mathtt{fei}$ to the more general case of arbitrary safety relations, we replace the functor $\mathtt{feiF}_\varphi : \mathcal{P}(\mathcal{S}_1 \times \mathcal{S}_2) \xrightarrow{mon} \mathcal{P}(\mathcal{S}_1 \times \mathcal{S}_2)$, with a new functor $\mathtt{feF}$ that has the following signature:

$$\mathtt{feF} : \mathcal{P}(\mathcal{S}_1 \times \mathcal{S}_2 \times \mathcal{P}(\mathcal{E}_1^\omega \times \mathcal{E}_2^\omega)) \xrightarrow{mon} \mathcal{P}(\mathcal{S}_1 \times \mathcal{S}_2 \times \mathcal{P}(\mathcal{E}_1^\omega \times \mathcal{E}_2^\omega))$$

The associated greatest fixed point $\mathtt{fe} \triangleq \nu.\mathtt{feF}$ relates two states and a trace relation. The intention is that for any safety relation ψ, a proof of $(s_1, s_2, \psi) \in \mathtt{fe}$ should guarantee that $(s_1, s_2) \in \forall^{TS_1}\exists^{TS_2}\psi$. Importantly, with this more general signature, the trace relation ψ can change over the course of a coinductive proof. This will allow us to use the notion of derivatives to *unroll* ψ. We start by defining an operator $\mathtt{next}^R_{e_1,e_2}(\psi) \subseteq \mathcal{S}_1 \times \mathcal{S}_2$:

$$\mathtt{next}^R_{e_1,e_2}(\psi) \triangleq \{ (s_1, s_2) \mid \Delta_{e_1,e_2}(\psi) \neq \emptyset \land (s_1, s_2, \Delta_{e_1,e_2}(\psi)) \in R \}$$

We use $\mathtt{next}$ to define the coinductive relation $\mathtt{fe}$:

$$\begin{aligned} \mathtt{feF}(R) &\triangleq \{ (s_1, s_2, \psi) \mid \forall s_1 \overset{e_1}{\rightsquigarrow} s_1', \exists s_2 \overset{e_2}{\rightsquigarrow} s_2', (s_1', s_2') \in \mathtt{next}^R_{e_1,e_2}(\psi) \} \\ \mathtt{fe} &\triangleq \nu.\mathtt{feF} \end{aligned}$$

Theorem 4.5 (Soundness of $\mathtt{fe}$). *Let ψ be a safety relation,*

$$(\mathcal{I}_1, \mathcal{I}_2, \psi) \in \mathtt{fe} \implies \models \forall^{TS_1}\exists^{TS_2}\psi$$

Proof. The proof follows exactly the same structure as for $\mathtt{fei}$, we just slightly change the abstract transition system. We extend the set of proof states to be $\Pi \triangleq \{ (s_1, s_2, \tau_1, \psi) \mid \tau_1 \in Traces(s_1) \land (s_1, s_2, \psi) \in \mathtt{fe} \}$ and the transition relation is defined as follows:

$$\frac{\tau_1 = e_1\tau_1' \qquad s_1 \overset{e_1}{\rightsquigarrow} s_1' \qquad s_2 \overset{e_2}{\rightsquigarrow} s_2' \qquad \Delta_{e_1,e_2}(\psi) \neq \emptyset}{(s_1, s_2, \tau_1, \psi) \xrightarrow{e_2}_{proof} (s_1', s_2', \tau_1', \Delta_{e_1,e_2}(\psi))}$$

Given two states and a safety relation ψ such that $(s_1, s_2, \psi) \in \mathtt{fe}$, and provided $\tau_1 \in Traces$, we can execute the abstract transition system from the initial proof state (s_1, s_2, τ_1) to obtain a trace τ_2. We pick τ_2 as a witness for the existential quantifier. It remains to prove that $(\tau_1, \tau_2) \in \psi$. Since ψ is a safety relation, by 4.3 it suffices to show $(\tau_1, \tau_2, \psi) \in \mathtt{safe}$. This fact is easily proven by coinduction (exploiting $(s_1, s_2, \psi) \in \mathtt{fe}$ and the fact that τ_2 is generated by executing the abstract transition system defined by $\rightarrow_{proof}$). □

As for `fei`, the definition of `fe` as a greatest fixed point immediately gives us the ability to perform incremental proofs by parameterized coinduction. To use `fe` in incremental proofs, we redefine our semantic quadruples using `feF` instead of `feiF` as the underlying functor:

$$\boxed{H} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\ \psi\ \} \triangleq (s_1, s_2, \psi) \in G_{\mathtt{feF}}(H)$$

$$\bbox[border:1px dashed]{H} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\ \psi\ \} \triangleq (s_1, s_2, \psi) \in H \cup G_{\mathtt{feF}}(H)$$

Note that here, ψ is not just a relation between events, but a relation between infinite traces. For convenience, we also introduce the following sextuple:

$$\boxed{H} \vdash e_1 \triangleright {s_1}^{\forall} \mid e_2 \triangleright {s_2}^{\exists} \; \{\ \psi\ \} \triangleq (s_1, s_2) \in \mathtt{next}_{e_1,e_2}^{H \,\cup\, G_{\mathtt{feF}}(H)}(\psi)$$

The corresponding reasoning rules are presented in Figure 3.

$$\textsc{Init}\quad \dfrac{\psi \text{ is a safety relation} \qquad \boxed{\emptyset} \vdash {\mathcal{I}_1}^{\forall} \mid {\mathcal{I}_2}^{\exists} \; \{\ \psi\ \}}{\models \forall^{TS_1}\exists^{TS_2}\psi} \qquad\qquad \textsc{Cycle}\quad \dfrac{(s_1, s_2, \psi) \in H}{\bbox[border:1px dashed]{H} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\ \psi\ \}}$$

$$\textsc{Step}\quad \dfrac{\forall s_1 \overset{e_1}{\rightsquigarrow} s_1', \exists s_2 \overset{e_2}{\rightsquigarrow} s_2', \ \boxed{H} \vdash e_1 \triangleright {s_1'}^{\forall} \mid e_2 \triangleright {s_2'}^{\exists} \; \{\ \psi\ \}}{H^{?} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\ \psi\ \}}$$

$$\textsc{Invariant}\quad \dfrac{(s_1, s_2, \psi) \in H' \qquad \forall (s_1, s_2, \psi) \in H', \ \boxed{H \cup H'} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\ \psi\ \}}{H^{?} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\ \psi\ \}}$$

$$\textsc{Deriv}\quad \dfrac{\Delta_{e_1,e_2}(\psi) \neq \emptyset \qquad \bbox[border:1px dashed]{H} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\ \Delta_{e_1,e_2}(\psi)\ \}}{\boxed{H} \vdash e_1 \triangleright {s_1}^{\forall} \mid e_2 \triangleright {s_2}^{\exists} \; \{\ \psi\ \}}$$

Fig. 3. Incremental reasoning rules for `fe`

The rules Init, Cycle, Step, and Invariant are similar to the ones of Figure 2. The main difference is that the Step rule *suspends* the proof temporarily by requiring us to check whether the choice of transition $s_2 \overset{e_2}{\rightsquigarrow} s_2'$ in reaction to a transition $s_1 \overset{e_1}{\rightsquigarrow} s_1'$ is *compatible* with the safety relation ψ. This is reflected by a premise of the form $\boxed{H} \vdash e_1 \triangleright {s_1}^{\forall} \mid e_2 \triangleright {s_2}^{\exists} \; \{\ \psi\ \}$ that can be discharged using the rule Deriv. The rule Deriv requires us to prove two things:

(1) ψ should not be immediately violated by the choice of event e_2; formalized by $\Delta_{e_1,e_2}(\psi) \neq \emptyset$
(2) ψ should not be violated later; formalized by $\bbox[border:1px dashed]{H} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \; \{\ \Delta_{e_1,e_2}(\psi)\ \}$

Note that the guard around H is only released after an application of Deriv. In addition to the rules presented in Figure 3, `fe` also inherits the Steps-L and Steps-R rules of `fei` (adapted to triples (s_1, s_2, ψ) in an obvious way).

4.4 Handling Derivatives

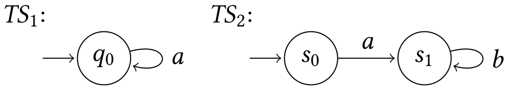
While all rules presented in Figure 3 are sound, they can be difficult to use in practice because of the *semantic* treatment of derivatives (as opposed to a more *syntactic* one). First, the Deriv rule requires us to prove the non-emptiness of a trace relation. Further, after a relation ψ has been derived, the proof focuses on a new relation $\Delta_{e_1,e_2}(\psi)$. There is *a priori* no reason to believe that $\Delta_{e_1,e_2}(\psi)$ has been encountered before, and therefore, it is not obvious that it will later be possible to use the rule Cycle to conclude the proof. Fortunately, when ψ is expressed using LTL modalities, there is a systematic way to determine an LTL representation of $\Delta_{e_1,e_2}(\psi)$ by recursing on the syntactic structure of the formula. Figure 4 describes how[1].

$$
\begin{aligned}
\Delta_{e_1,e_2}(\Box\psi) &\equiv \Delta_{e_1,e_2}(\psi) \wedge \Box\psi \\
\Delta_{e_1,e_2}(\psi_1 \wedge \psi_2) &\equiv \Delta_{e_1,e_2}(\psi_1) \wedge \Delta_{e_1,e_2}(\psi_2) \\
\Delta_{e_1,e_2}(\psi_1 \vee \psi_2) &\equiv \Delta_{e_1,e_2}(\psi_1) \vee \Delta_{e_1,e_2}(\psi_2) \\
\Delta_{e_1,e_2}(\psi_1 \mathbin{\mathtt{W}} \psi_2) &\equiv \Delta_{e_1,e_2}(\psi_2) \vee [\Delta_{e_1,e_2}(\psi_1) \wedge (\psi_1 \mathbin{\mathtt{W}} \psi_2)] \\
\Delta_{e_1,e_2}(\bigcirc\psi) &\equiv \psi \\
\Delta_{e_1,e_2}(\varphi) &\equiv \begin{cases} \mathit{true} & \text{if } (e_1, e_1) \in \varphi \\ \mathit{false} & \text{otherwise} \end{cases}
\end{aligned}
$$

Fig. 4. Derivatives of common temporal modalities

We note that for temporal modalities $\Box$ and $\mathtt{W}$, the original formula occurs as a subformula in the derivative. In a sense, the Δ operator *reveals* the coinductive nature of these modalities. This will help us closing cycles in proofs.

Example 4.6. We consider the following transition systems:



We wish to prove $\models \forall^{TS_1}\exists^{TS_2}(a_1 \wedge a_2) \mathbin{\mathtt{W}} b_2$. We start by applying Init, followed by Step to match the a-transition in TS_1 with an a-transition in TS_2.

$$
\begin{aligned}
& \models \forall^{TS_1}\exists^{TS_2}(a_1 \wedge a_2) \mathbin{\mathtt{W}} b_2 \\
\Longleftarrow\quad & \boxed{\emptyset} \vdash {q_0}^{\forall} \mid {s_0}^{\exists} \ \{\ (a_1 \wedge a_2) \mathbin{\mathtt{W}} b_2\ \} && \text{(by Init)} \\
\Longleftarrow\quad & \boxed{\emptyset} \vdash a \triangleright {q_0}^{\forall} \mid a \triangleright {s_1}^{\exists} \ \{\ (a_1 \wedge a_2) \mathbin{\mathtt{W}} b_2\ \} && \text{(by Step)}
\end{aligned}
$$

To make further progress, we need to prove that emitting two a's does not immediately violate the current temporal relation. To do so, we apply the rule Deriv and we obtain

$$\Delta_{a,a}((a_1 \wedge a_2) \mathbin{\mathtt{W}} b_2) \neq \emptyset \wedge \bbox[border:1px dashed]{\emptyset} \vdash {q_0}^{\forall} \mid {s_1}^{\exists} \ \{\ \Delta_{e_1,e_2}((a_1 \wedge a_2) \mathbin{\mathtt{W}} b_2)\ \} \quad \text{(by Deriv)}$$

Using known derivatives (Figure 4), we can easily show that $\Delta_{a,a}((a_1 \wedge a_2) \mathbin{\mathtt{W}} b_2) \equiv (a_1 \wedge a_2) \mathbin{\mathtt{W}} b_2 \neq \emptyset$. After substituting the result of the derivative we are therefore left with the goal

$$\bbox[border:1px dashed]{\emptyset} \vdash {q_0}^{\forall} \mid {s_1}^{\exists} \ \{\ (a_1 \wedge a_2) \mathbin{\mathtt{W}} b_2\ \}$$

By Step we match an a-transition in TS_1 with a b-transition in TS_2 and we obtain

$$\boxed{\emptyset} \vdash a \triangleright {q_0}^{\forall} \mid b \triangleright {s_1}^{\exists} \ \{\ (a_1 \wedge a_2) \mathbin{\mathtt{W}} b_2\ \} \quad \text{(by Step)}$$

[1] To the best of our knowledge, the notion of derivatives has never explicitly been formalized for LTL-defined languages. However, derivatives have been applied to languages over infinite words expressed as omega-regular expressions [39]. Further, derivatives are easily recovered from the expansion laws of temporal modalities [3].

Again, by Deriv, we have to prove two goals:

$$\Delta_{a,b}((a_1 \land a_2) \mathbin{\mathsf{W}} b_2) \neq \emptyset \land \begin{array}{:c:}\hdashline \emptyset \\ \hdashline \end{array} \vdash {q_0}^{\forall} \mid {s_1}^{\exists} \ \{ \ \Delta_{e_1,e_2}((a_1 \land a_2) \mathbin{\mathsf{W}} b_2) \ \} \quad \text{(by Deriv)}$$

Since $\Delta_{a,b}((a_1 \land a_2) \mathbin{\mathsf{W}} b_2) \equiv \mathit{true} \neq \emptyset$, it suffices to prove the following:

$$\begin{array}{:c:}\hdashline \emptyset \\ \hdashline \end{array} \vdash {q_0}^{\forall} \mid {s_1}^{\exists} \ \{ \ \mathit{true} \ \}$$

Using Invariant we set the current coinduction hypothesis to be $H \triangleq \{ \ (q_0, s_1, \mathit{true}) \ \}$

$$\boxed{H} \vdash {q_0}^{\forall} \mid {s_1}^{\exists} \ \{ \ \mathit{true} \ \} \quad \text{(by Invariant)}$$

By Step we again match an a-transition with a b-transition and we get

$$\boxed{H} \vdash a \triangleright {q_0}^{\forall} \mid b \triangleright {s_1}^{\exists} \ \{ \ \mathit{true} \ \} \quad \text{(by Step)}$$

Since the derivative of *true* is *true* for any pair of events, by Deriv we obtain

$$\begin{array}{:c:}\hdashline H \\ \hdashline \end{array} \vdash {q_0}^{\forall} \mid {s_1}^{\exists} \ \{ \ \mathit{true} \ \} \quad \text{(by Deriv)}$$

Finally, since $(q_0, s_1, \mathit{true}) \in H$, we can conclude using Cycle. □

While it is correct, the proof presented above suffers from several limitations.

Limitation 1. Even though we purposely gave a very detailed proof, the example discussed above still seems abnormally long. In particular, the last application of the rule Step seems redundant. Indeed, if we look at the structure of the two transition systems, the proof should only have two easy steps: one to match the a-transition with an a-transition, and a second one to match an a-transition with a b-transition. More precisely, once we reached the goal $\boxed{\emptyset} \vdash {q_0}^{\forall} \mid {s_1}^{\exists} \ \{ \ \mathit{true} \ \}$, we would like to be able to conclude right away. This goal should, in principle, immediately follow from the previous and intuitively more difficult goal $\boxed{\emptyset} \vdash {q_0}^{\forall} \mid {s_1}^{\exists} \ \{ \ (a_1 \land a_2) \mathbin{\mathsf{W}} b_2 \ \}$.

Limitation 2. Another and more subtle concern is the fact that whenever we applied Deriv, we had to explicitly compute the derivative of the current trace relation, and substitute the result back in the current goal. Substituting a relation with an equivalent one is not an issue when doing informal reasoning in set theory; however, in type-theory based proof assistants such as Coq, this is not possible in general. By default, we can only substitute a relation for another one if both relations are equal in the sense of propositional equality. One solution would be to assume the axiom of *predicate extensionality* that precisely stipulates that two equivalent predicates are also equal.

Axiom 1 (Predicate Extensionality). Let $P, Q \in \mathcal{P}(A)$ be predicates over A, then $P \equiv Q \implies P = Q$.

Note however that this approach is not entirely satisfactory as it would make our development depends on a second axiom in addition to functional choice.

4.5 Strengthening Is All We Need

Instead of relying on additional axioms, a more general solution to address the two limitations presented in the previous section is to prove once and for all that our coinductive relation supports the following *strengthening* rule:

$$\textsc{Strengthen}\ \frac{\psi' \subseteq \psi \qquad H^{?} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \ \{ \ \psi' \ \}}{H^{?} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \ \{ \ \psi \ \}}$$

Note that the two question marks in Strengthen should be interpreted as follows: the rule can be applied both when H is guarded or unguarded, but it never modifies the guard (in particular, if the hypothesis was guarded, it remains guarded). The addition of Strengthen would clearly solve the first limitation we discussed: it can factor redundant proof steps by reusing results already established for stronger trace relations. Further, it also solves the substitution problem. Indeed, substituting a trace relation with an equivalent one is just a special case of strengthening.

Unfortunately, with our current definition of semantic quadruples, Strengthen is unsound. To understand why, let us first observe that the following weaker version of the rule *is* sound:

$$\frac{\psi' \subseteq \psi \qquad \boxed{\emptyset} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \psi'\ \}}{H^{?} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \psi\ \}}$$

Note that in this variant, H is replaced by $\emptyset$ after applying the rule! In other words, if we need to prove $H^{?} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \psi\ \}$, it suffices to prove that the property holds for a stronger trace relation $\psi' \subseteq \psi$, but at the cost of forgetting all knowledge previously accumulated in the proof. This essentially ruins the benefits of having an incremental proof system. Intuitively the reason why we need to reset H is that the triples we explore during the proof of $\boxed{H} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \psi'\ \}$ are not necessarily the same as the triples explored during a proof of $\boxed{H} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \psi\ \}$. In particular, instead of considering the successive derivatives of ψ, we are instead considering the successive derivatives of ψ'. In turn, if we use H to reason about triples emanating from (s_1, s_2, ψ'), there is no reason to believe that the same reasoning carries over to triples emanating from (s_1, s_2, ψ) without further assumptions. Fortunately, we can slightly generalize our coinductive relation in such a way that Strengthen (in its more general version) is provably sound, and all other proof rules remain valid as well. The next section discusses this generalization.

For now, let us assume that Strengthen is sound. Using the equivalences listed in Figure 4, Strengthen can be combined with Deriv to obtain a set of rules for handling temporal modalities. Figure 5 presents a selection of such rules.

Deriv-□

$$\frac{(e_1, e_2) \in \varphi \qquad \dbox{H} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \Box\varphi\ \}}{\boxed{H} \vdash e_1 \triangleright s_1{}^{\forall} \mid e_2 \triangleright s_2{}^{\exists}\ \{\ \Box\varphi\ \}}$$

Deriv-○

$$\frac{\psi \neq \emptyset \qquad \dbox{H} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \psi\ \}}{\boxed{H} \vdash e_1 \triangleright s_1{}^{\forall} \mid e_2 \triangleright s_2{}^{\exists}\ \{\ \bigcirc\psi\ \}}$$

Deriv-W-Now

$$\frac{(e_1, e_2) \in \varphi_2 \qquad \dbox{H} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \mathit{true}\ \}}{\boxed{H} \vdash e_1 \triangleright s_1{}^{\forall} \mid e_2 \triangleright s_2{}^{\exists}\ \{\ \varphi_1 \mathsf{W} \varphi_2\ \}}$$

Deriv-W-Later

$$\frac{(e_1, e_2) \in \varphi_1 \qquad \dbox{H} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \varphi_1 \mathsf{W} \varphi_2\ \}}{\boxed{H} \vdash e_1 \triangleright s_1{}^{\forall} \mid e_2 \triangleright s_2{}^{\exists}\ \{\ \varphi_1 \mathsf{W} \varphi_2\ \}}$$

Fig. 5. Selection of proof rules for temporal modalities

5 Up-to Techniques

5.1 Reasoning Up-to Stronger Trace Relations

Often times, we have a goal of the form $\dbox{H} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \psi\ \}$ where the triple (s_1, s_2, ψ) *almost* belongs to H (which would allow us to conclude using the rule Cycle), but not *exactly*. In other words, we would like to be able to reason *up to* slight enlargements of the coinduction hypothesis [36]. As discussed in the previous section, a typical occurrence of this pattern is when we have a goal of the form $\dbox{H} \vdash s_1{}^{\forall} \mid s_2{}^{\exists}\ \{\ \psi\ \}$ but H only contains (s_1, s_2, ψ') for some stronger relation $\psi' \subset \psi$. In [34] and [35], Pous developed systematic ways to soundly integrate up-to techniques in coinductive proofs. Building on these ideas, Zakowski et al. later showed that the parameterized greatest fixed point operator G can be generalized to support up-to techniques as well as other enhancements [43]. This generalization, usually referred to as GPACO, preserves the ability to perform incremental proofs. The reasoning rules of GPACO are presented in Figure 6. We note that the original definition by Zakowski et al. is more involved, and we refer the readers to [43] for a more in-depth presentation. However, our Coq development relies on an instance of GPACO which exposes exactly the simplified interface described in Figure 6. Contrary to standard parameterized coinduction, GPACO explicitly marks when the current assumption can be used or not: we note $\boxed{G_F(H)}$ when the assumption H is guarded, and $\dbox{G_F(H)}$ when it is unguarded. When proof rules can be applied regardless of whether the coinduction hypothesis is guarded or not, we note $G^?_F(H)$.

$$\textsc{Init}\ \frac{X \subseteq \boxed{G_F(\emptyset)}}{X \subseteq \nu.F} \qquad \textsc{Step}\ \frac{X \subseteq F(\dbox{G_F(H)})}{X \subseteq G^?_F(H)} \qquad \textsc{Accumulate}\ \frac{X \subseteq \boxed{G_F(X \cup H)}}{X \subseteq G^?_F(H)}$$

$$\textsc{Cycle}\ \frac{X \subseteq H}{X \subseteq \dbox{G_F(H)}} \qquad \textsc{Up-to}\ \frac{\mathit{clo} \text{ is } \mathit{compatible} \text{ with } F}{\mathit{clo}(G^?_F(H)) \subseteq G^?_F(H)}$$

Fig. 6. Rules of GPACO

As for parameterized coinduction (Figure 1), the rule Step releases the guard. When the current hypothesis is unguarded, the rule Cycle can be invoked to conclude a proof. Finally, the rule Up-to enables up-to reasoning. The idea is that given an operator *clo* that satisfies a certain compatibility criterion (Formally, *clo* is *compatible* with F iff $\mathit{clo} \circ F \subseteq F \circ \mathit{clo}$ [35]), the goal $X \subseteq G^?(H)$ can be replaced by the hopefully simpler goal $X \subseteq \mathit{clo}(G^?(H))$. Typically, we choose *clo* to be a closure operator (hence the naming convention *clo*) so that $\mathit{clo}(G^?(H))$ is larger than $G^?(H)$. To support up-to techniques, we redefine our semantic tuples using GPACO as follows:

$$\boxed{H} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \{ \psi \} \triangleq (s_1, s_2, \psi) \in \boxed{G_{\mathtt{feF}}(H)}$$

$$\dbox{H} \vdash {s_1}^{\forall} \mid {s_2}^{\exists} \{ \psi \} \triangleq (s_1, s_2, \psi) \in \dbox{G_{\mathtt{feF}}(H)}$$

$$\boxed{H} \vdash e_1 \triangleright {s_1}^{\forall} \mid e_2 \triangleright {s_2}^{\exists} \{ \psi \} \triangleq (s_1, s_2) \in \mathtt{next}^{\dbox{G_{\mathtt{feF}}(H)}}_{e_1,e_2}(\psi)$$

All the reasoning rules previously introduced are preserved by this change. However, this generalization naturally inherits the Up-to rule. In particular, it allows us to reason up to stronger trace relations. The associated closure operator `strclo` is defined as follows:

$$\mathtt{strclo}(H) \triangleq \{ (s_1, s_2, \psi) \mid \exists \psi' \subseteq \psi, (s_1, s_2, \psi') \in H \}$$

By definition, to prove that a triple (s_1, s_2, ψ) belongs to $\mathtt{strclo}(H)$, it suffices to prove that (s_1, s_2, ψ') is in H for some relation $\psi' \subseteq \psi$. Importantly, we observe that `strclo` is compatible with the monotone functor `feF`. Consequently, Strengthen is just an instance of Up-to with `feF` as the underlying functor, and `strclo` as the (compatible) closure operator.

5.2 Reasoning Up-to Simulation

To prove a universal trace property $\forall^{TS}\psi$, it suffices to show that the property hold for any superset $T \supseteq \mathit{Traces}(TS)$. Dually, to prove an existential trace property $\exists^{TS}\psi$, it suffices to show that the property holds for a subset $T \subseteq \mathit{Traces}(TS)$. In turn, to prove a $\forall\exists$ hyperproperty, it suffices to consider an over-approximation of the universally quantified traces and an under-approximation of the existentially quantified traces. In some cases, reasoning about approximated sets of traces can greatly simplify proofs. This intuition about traces can be lifted at the level of states if we consider simulation instead of trace inclusion. In particular, our coinductive relation can be equipped with the following sound rules:

$$\textsc{Sim-L}\quad \frac{(s_1, s_1') \in \mathtt{sim} \qquad H^? \vdash {s_1'}^{\forall} \mid {s_2}^{\exists}\ \{\ \psi\ \}}{H^? \vdash {s_1}^{\forall} \mid {s_2}^{\exists}\ \{\ \psi\ \}} \qquad\qquad \textsc{Sim-R}\quad \frac{(s_2', s_2) \in \mathtt{sim} \qquad H^? \vdash {s_1}^{\forall} \mid {s_2'}^{\exists}\ \{\ \psi\ \}}{H^? \vdash {s_1}^{\forall} \mid {s_2}^{\exists}\ \{\ \psi\ \}}$$

These rules can be used at any point in a proof to replace a state with a similar state of our choice. The soundness of Sim-L and Sim-R is immediately obtained from Up-to by reasoning up to *simulation*. We define the corresponding (compatible) closure operator as follows:

$$\mathtt{simclo}(H) \triangleq \{\ (s_1, s_2) \mid \exists (s_1', s_2'), (s_1, s_1') \in \mathtt{sim} \wedge (s_2', s_2) \in \mathtt{sim} \wedge (s_1', s_2') \in H\ \}$$

We refer the readers to our Coq development for a proof that simclo is compatible with feF. Since simulation is just a special instance of fe with $\psi = \square\, eq$, we note that the simulation rules can be reformulated purely in terms of our semantic quadruples as follows:

$$\textsc{Sim-L (internalized)}\quad \frac{\boxed{\emptyset} \vdash {s_1}^{\forall} \mid {s_1'}^{\exists}\ \{\ \square\, eq\ \} \qquad H^? \vdash {s_1'}^{\forall} \mid {s_2}^{\exists}\ \{\ \psi\ \}}{H^? \vdash {s_1}^{\forall} \mid {s_2}^{\exists}\ \{\ \psi\ \}} \qquad \textsc{Sim-R (internalized)}\quad \frac{\boxed{\emptyset} \vdash {s_2'}^{\forall} \mid {s_2}^{\exists}\ \{\ \square\, eq\ \} \qquad H^? \vdash {s_1}^{\forall} \mid {s_2'}^{\exists}\ \{\ \psi\ \}}{H^? \vdash {s_1}^{\forall} \mid {s_2}^{\exists}\ \{\ \psi\ \}}$$

6 From $\forall\exists$ to $\forall^*\exists^*$

In the previous sections, we focused exclusively on hyperproperties with one universal quantifier followed by one existential quantifier. We note that our coinductive relations can naturally be extended to support the more general case of hyperproperties of the form $\forall^*\exists^*$ (i.e., 0 or more universal quantifiers followed by 0 or more existential quantifiers). Given a family of $m + n$ transition systems $TS_1, \ldots, TS_{n+m}$ such that $TS_i = (\mathcal{S}_i, \mathcal{E}_i, \mathcal{I}_i, \rightarrow_i)$ for all $0 \leq i \leq n + m$, we define the coinductive relation $\mathsf{hyco}^{m,n} \subseteq \mathcal{S}_1 \times \ldots \times \mathcal{S}_{n+m} \times \mathcal{P}(\mathcal{E}_1^{\omega} \times \ldots \times \mathcal{E}_{n+m}^{\omega})$ as follows:

$$\begin{aligned}\mathsf{hycoF}^{m,n}(R) \triangleq \{\ & (s_1, \ldots s_m, s_{m+1}, \ldots, s_{m+n}) \mid \\ & \forall s_1 \overset{e_1}{\rightsquigarrow} s_1' \ldots \forall s_m \overset{e_m}{\rightsquigarrow} s_m', \exists s_{m+1} \overset{e_{m+1}}{\rightsquigarrow} s_{m+1}' \ldots \exists s_{m+n} \overset{e_{m+n}}{\rightsquigarrow} s_{m+n}', \\ & \Delta_{e_1,\ldots,e_{n+m}}(\psi) \neq \emptyset \wedge (s_1', \ldots, s_{m+n}', \Delta_{e_1,\ldots,e_{n+m}}(\psi)) \in R\ \}\end{aligned}$$

The associated greatest fixed point $\mathsf{hyco}^{m,n} \triangleq \nu.\mathsf{hycoF}^{m,n}$ gives a sound proof technique for $\forall^*\exists^*$ hyperproperties.

Theorem 6.1. *Let $\psi \subseteq \mathcal{P}(\mathcal{E}_1^{\omega} \times \ldots \times \mathcal{E}_{n+m}^{\omega})$ be a $(m + n)$-ary safety relation. Then,*

$$(\mathcal{I}_1, \ldots, \mathcal{I}_{m+n}, \psi) \in \mathsf{hyco}^{m,n} \implies \models \forall^{TS_1} \ldots \forall^{TS_m} \exists^{TS_{m+1}} \ldots \exists^{TS_{m+n}} \psi$$

7 Proving Temporal Hyperproperties of Imperative Programs with I/O

So far, we have introduced a general framework to prove $\forall\exists$ hyperproperties with a safety relation between the universal and existential traces. In this section, we show that this framework naturally applies to the verification of imperative programs with inputs, outputs, and nondeterminism (IMP$_{io}$).

7.1 An Imperative Programming Language with Nondeterminism and I/O

IMP$_{io}$ programs support infinite loops, inputs, outputs, conditionals, and nondeterministic generation of constant values. The syntax is described in Figure 7.

$$P ::= \textbf{loop}\ P \mid \textbf{if}\ c\ \textbf{then}\ P\ \textbf{else}\ P \mid \textbf{input}\ x \mid \textbf{output}\ e \mid \textbf{havoc}\ x \mid P\ \textbf{;}\ P \mid x \mathrel{\textbf{:=}} e$$

$$e ::= x \mid e + e \mid e \text{ - } e \mid 0 \mid 1 \mid 2 \mid \ldots \qquad c ::= \textbf{true} \mid \textbf{false} \mid \mathrm{e} < e \mid e = e \mid \ldots$$

Fig. 7. Syntax of IMP$_{io}$

In Figure 7, x denotes a variable drawn from a set of program variables $\mathcal{X}$, P denotes a program, e an arithmetic expression, and c a boolean expression. Given a memory $m : \mathcal{X} \to \mathbb{Z}$, we denote by $[\![e]\!]_m \in \mathbb{Z}$ (resp. $[\![c]\!]_m \in \{\mathit{true}, \mathit{false}\}$) the value of arithmetic (resp. boolean) expressions. We interpret IMP$_{io}$ programs as labeled transition systems by giving a small-step operational semantics which, in turn, naturally induces a trace semantics.

$$\textsc{Loop}\ \frac{}{\langle \textbf{loop}\ P, m\rangle \longrightarrow \langle P; \textbf{loop}\ P, m\rangle} \qquad \textsc{Continue}\ \frac{}{\langle (P_1\ ;\ P_2)\ ;\ P_3, m\rangle \longrightarrow \langle P_1\ ;\ (P_2\ ;\ P_3), m\rangle}$$

$$\textsc{Input}\ \frac{v \in \mathbb{Z}}{\langle \textbf{input}\ x\ ;\ P, m\rangle \xrightarrow{\mathtt{in}(v)} \langle P, m[x \leftarrow v]\rangle} \qquad \textsc{Output}\ \frac{v = [\![e]\!]_m}{\langle \textbf{output}\ e\ ;\ P, m\rangle \xrightarrow{\mathtt{out}(v)} \langle P, m\rangle}$$

$$\textsc{Havoc}\ \frac{v \in \mathbb{Z}}{\langle \textbf{havoc}\ x\ ;\ P, m\rangle \longrightarrow \langle P, m[x \leftarrow v]\rangle} \qquad \textsc{Assign}\ \frac{v = [\![e]\!]_m}{\langle x \mathrel{\textbf{:=}} e\ ;\ P, m\rangle \longrightarrow \langle P, m[x \leftarrow v]\rangle}$$

$$\textsc{Ite-true}\ \frac{[\![c]\!]_m = \textbf{true}}{\langle \textbf{if}\ c\ \textbf{then}\ P_1\ \textbf{else}\ P_2, m\rangle \longrightarrow \langle P_1, m\rangle} \qquad \textsc{Ite-false}\ \frac{[\![c]\!]_m = \textbf{false}}{\langle \textbf{if}\ c\ \textbf{then}\ P_1\ \textbf{else}\ P_2, m\rangle \longrightarrow \langle P_2, m\rangle}$$

Fig. 8. Operational semantics of IMP$_{io}$

We note that **havoc** and **input** have the same effect on the memory, but **input** effects are recorded in program traces, whereas **havoc** is used to model silent nondeterministic assignments and does not emit any event. Further, since we are considering only *truly* reactive programs, we do not assign a meaning to basic instructions when they are not followed by more instructions (see rules Input, Output, Havoc, and Assign). Such programs would not have any infinite trace anyway.

7.2 A Deductive System for Hyperproperties of Imperative Programs

Let $\mathbb{P}$ be the set of all programs, $\mathcal{S} \triangleq \mathbb{P} \times \mathbb{Z}^{\mathcal{X}}$ be the set of all program states (pairs of a program and a memory), and $\mathcal{E} \triangleq \{\mathtt{in}(x) \mid x \in \mathbb{Z}\} \cup \{\mathtt{out}(x) \mid x \in \mathbb{Z}\}$ be the set of possible events emitted by *IMP$_{io}$* programs. Any program P can be viewed as labeled transition system $TS_P \triangleq (\mathcal{S}, \mathcal{E}, \mathcal{I}_P, \longrightarrow)$ where $\longrightarrow$ is the small-step operational semantics of IMP$_{io}$, and $\mathcal{I}_P \triangleq \langle P, \_ \mapsto 0\rangle$ is the state initiating the execution of P in a zero-initialized memory. In the following, we will write P for TS_P. In this setting, we can immediately use our coinductive relation `fe` to verify hyperproperties of programs. Nonetheless, instead of reasoning with the high-level rules presented in the previous sections, we can exploit the syntactic structure of programs to obtain more convenient reasoning rules as well as dedicated proof tactics. Figure 9 presents a selection of such reasoning rules. Many more can be derived.

Proof Management. The first rule, Init, exploits the soundness of `fe` (Theorem 4.5) to initialize a proof by parameterized coinduction. The second rule, Memory-Invariant, is derived from Invariant and allows to extend the current coinduction hypothesis with a relation that should hold for every pairs of memories from the point of application on. Importantly, contrary to the Invariant rule (see Figure 3), where the extension of the coinduction hypothesis refers to states of the underlying LTSs, Memory-Invariant is restricted to invariants that only refer to the memories of the two programs (ignoring the code of the programs themselves, and the current temporal relation ψ). This is not a hard restriction (we could still use the more general rule), but we adopt this version for convenience as it is relatively unusual to allow invariants to refer to the syntactic structure of a program or to a logical specification. A memory-invariant *INV* is lifted to a relation on triples in a natural way. More concretely, we define

$$INV@(P_1, P_2, \psi) \triangleq \{ \, (\langle P_1, m_1 \rangle, \langle P_2, m_2 \rangle, \psi) \mid (m_1, m_2) \in INV \, \}$$

The rule Memory-Invariant extends the current coinduction hypothesis with $INV@(P_1, P_2, \psi)$, allowing to finish a proof at a later point if we cycle back to a state where the same pair of programs has to be matched with the same temporal relation ψ.

Handling I/O. To handle inputs and outputs, we introduce two specialized rules Input-input and output-output. These rules are exploiting the fact that if the next instruction to be executed in both programs is an I/O operation, the only way to make progress is to ensure that any event produced by the left-hand program can be matched with a corresponding event in the right-hand program. Once the appropriate event for the right-hand program has been chosen, the new goal is guarded by a `next` operator, forcing us to compute the derivative of the current property ψ and prove that it is not trivially violated.

Handling Nondeterminism. Nondeterministic assignments are covered by the rules Havoc-L and Havoc-R. When the next instruction to execute in the left-hand program is a nondeterministic assignment of x, Havoc-L requires us to consider an arbitrary new value v for x (i.e., the new goal is guarded by a universal quantification over v). Dually, the rule Havoc-R handles nondeterministic assignments in the right-hand program by requiring us to *choose* a new value v for x (i.e., the new goal is guarded by an existential quantification over v). The rules Havoc-R and Havoc-L are derived from the more general Steps-L and Steps-R and from the operational semantics of IMP$_{io}$.

Handling Loops. To handle loops, one could simply use the rules Steps-L and Steps-R to unfold a loop either in the right-hand or the left-hand program. However, we almost always wish to establish an invariant when a loop is encountered. The rules Loop-L and Loop-R combine an application of Steps-L (or Steps-R) with an application of Memory-Invariant to simultaneously unfold loops and establish a new memory-invariant.

Handling Derivatives. We note that after applying I/O rules, the new goals are always of the form $\boxed{H} \vdash \ldots \triangleright \ldots^{\forall} \mid \ldots \triangleright \ldots^{\exists} \{ \, \psi \, \}$, forcing us to derive the current relation ψ to check that the emitted events are not immediately violating it. To do so, we use the rules for derivatives introduced in Figure 5.

Proof Management

$$\textsc{Init}\qquad \dfrac{\psi \text{ is a safety relation} \qquad \boxed{\emptyset} \vdash \mathcal{I}_{P_1}{}^{\forall} \mid \mathcal{I}_{P_2}{}^{\exists}\ \{\ \psi\ \}}{\models \forall^{P_1}\exists^{P_2}\psi}$$

$$\textsc{Memory-Invariant}\qquad \dfrac{(m_1, m_2) \in \mathit{INV} \qquad \forall (m_1, m_2) \in \mathit{INV}, \boxed{H \cup \mathit{INV}@(P_1, P_2, \psi)} \vdash \langle P_1, m_1\rangle^{\forall} \mid \langle P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}}{\boxed{H} \vdash \langle P_1, m_1\rangle^{\forall} \mid \langle P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}}$$

I/O

$$\textsc{Input-input}\qquad \dfrac{\forall v_1, \exists v_2, \boxed{H} \vdash \mathtt{in}(v_1) \triangleright \langle P_1, m_1[x_1 \mapsto v_1]\rangle^{\forall} \mid \mathtt{in}(v_2) \triangleright \langle P_2, m_2[x_2 \mapsto v_2]\rangle^{\exists}\ \{\ \psi\ \}}{\boxed{H} \vdash \langle \textbf{input}\ x_1\ ;\ P_1, m_1\rangle^{\forall} \mid \langle \textbf{input}\ x_2\ ;\ P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}}$$

$$\textsc{Output-output}\qquad \dfrac{\boxed{H} \vdash \mathtt{out}(\llbracket e_1 \rrbracket_{m_1}) \triangleright \langle P_1, m_1\rangle^{\forall} \mid \mathtt{out}(\llbracket e_2 \rrbracket_{m_2}) \triangleright \langle P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}}{\boxed{H} \vdash \langle \textbf{output}\ e_1\ ;\ P_1, m_1\rangle^{\forall} \mid \langle \textbf{output}\ e_2\ ;\ P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}}$$

Nondeterminism

$$\textsc{Havoc-L}\qquad \dfrac{\forall v, \boxed{H} \vdash \langle P_1, m_1[x \mapsto v]\rangle^{\forall} \mid \langle P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}}{\boxed{H} \vdash \langle \textbf{havoc}\ x\ ;\ P_1, m_1\rangle^{\forall} \mid \langle P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}}$$

$$\textsc{Havoc-R}\qquad \dfrac{\exists v, \boxed{H} \vdash \langle P_1, m_1\rangle^{\forall} \mid \langle P_2, m_2[x \mapsto v]\rangle^{\exists}\ \{\ \psi\ \}}{\boxed{H} \vdash \langle P_1, m_1\rangle^{\forall} \mid \langle \textbf{havoc}\ x\ ;\ P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}}$$

Loops

$$\textsc{Loop-L}\qquad \dfrac{\begin{array}{c}(m_1, m_2) \in \mathit{INV}\\ \forall (m_1, m_2) \in \mathit{INV}, \boxed{H \cup \mathit{INV}@(\textbf{loop}\ P_1, P_2, \psi)} \vdash \langle P_1\ ;\ \textbf{loop}\ P_1, m_1\rangle^{\forall} \mid \langle P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}\end{array}}{\boxed{H} \vdash \langle \textbf{loop}\ P_1, m_1\rangle^{\forall} \mid \langle P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}}$$

$$\textsc{Loop-R}\qquad \dfrac{\begin{array}{c}(m_1, m_2) \in \mathit{INV}\\ \forall (m_1, m_2) \in \mathit{INV}, \boxed{H \cup \mathit{INV}@(P_1, \textbf{loop}\ P_2, \psi)} \vdash \langle P_1\ ;\ \textbf{loop}\ P_1, m_1\rangle^{\forall} \mid \langle P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}\end{array}}{\boxed{H} \vdash \langle \textbf{loop}\ P_1, m_1\rangle^{\forall} \mid \langle P_2, m_2\rangle^{\exists}\ \{\ \psi\ \}}$$

Fig. 9. Selection of proof rules for IMP_{io}

7.3 Examples in Coq

Example 7.1. Consider the following example of a simple echo server:

```
        loop
echo:     input $x$
          output $x$
```

We use HyCo to prove that for all executions of echo, there exists another execution such that the outputs of the second execution are always exactly the double of the outputs in the first execution. The inputs are never compared. Formally, we prove $\forall^{\texttt{echo}}\exists^{\texttt{echo}} \Box\, \mathit{double}$ where $\mathit{double} \triangleq \{\ (e_1, e_2) \mid e_1 = \texttt{out}(x) \implies e_2 = \texttt{out}(2 * x)\ \}$. The proof script (slightly simplified for readability) corresponding to the proof of this example is the following:

```
Proof.
  hyco_init. hyco_sync. hyco_step.
  intros v1. exists (2 * v1).
  hyco_deriv. hyco_step.
    ...
  hyco_cycle.
Qed.
```

In this script, the tactic hyco_init exploits the rule Init to initialize a proof by coinduction. It also implicitly uses the rule Invariant to add the initial states to the current coinduction hypothesis. The tactic hyco_sync then steps through the execution of the two programs until both reach an I/O instruction. In our case, the programs are executed until their first **input** instruction. The tactic hyco_step determines which I/O rule should be applied to make further progress in the coinductive proof. Gere, Input-input is selected and it remains to match any input v_1 with some input v_2. We pick $v_2 = 2 * v_1$. After choosing an appropriate input, we need to check that this choice is compatible with the current trace relation by computing its derivative. The tactic hyco_deriv selects the appropriate derivation rule (see Figure 5). Here, the rule Deriv-$\Box$ is applied and requires us to prove that the two outputs $\texttt{out}(v_1)$ and $\texttt{out}(2 * v_1)$ are satisfying the event-invariant *double*. This fact is trivially proven by unfolding the definition of *double* and reading the current content of the memories. Finally, the tactic hyco_cycle terminates the proof by applying the coinduction hypothesis. Indeed, after completing one iteration of the loop, the two programs remaining to be executed are again two copies of echo, and the property we need to verify is still $\Box\, \mathit{double}$; we cycled back to the initial state of the proof. We note that the repeated use of the tactics hyco_sync and hyco_step could be partially automated, allowing us to focus only on the *interesting* part of the proof: picking the appropriate input for the right-hand program.

Example 7.2. We describe a second example that requires us to use the rule Invariant as well as the alignment rules discussed in Section 3.3. We consider the two following programs incr and ndet_add:

```
                                  $x$ := 0
      $x$ := 0                    loop
      loop                          havoc $y$
incr:   $x$ := $x$ + 1   ndet_add:  $x$ := $x$ + $y$
        output $x$                  output $x$
```

As for the previous example, we wish to verify $\forall^{\texttt{incr}}\exists^{\texttt{ndet\_add}} \Box\, \mathit{double}$. The corresponding proof script is presented below.

```
Proof.
  hyco_init.
  hyco_left 2. hyco_right 2.
  hyco_invariant (fun m1 m2 => 2 * m1 "x" = m2 "x").
    ...
  hyco_left 3. hyco_right 3.
  apply (hyco_havoc_r 2). hyco_right 2.
  hyco_step.
  ... rewrite <- INV. ...
  hyco_cycle.
  ... rewrite <- INV. ...
Qed.
```

The proof starts by executing both programs until the beginning of the loop is reached. To do so, we use the tactics `hyco_left` and `hyco_right`. These tactics exploit the rules Steps-L and Steps-R (see section 3.3) to perform n silent computation steps in the left-hand or the right-hand program. Once the beginning of both loops is reached, we use the tactic `hyco_invariant` to establish the memory-invariant $x_2 = 2 * x_1$. After proving that the invariant is satisfied initially, we again step through the programs and we resolve the nondeterministic assignment **havoc** y from the right-hand program by choosing $y = 2$. Note that at this point of the proof, the left-hand program is ready to emit an output while the right-hand program still needs to update its variable x. We use the tactic `hyco_right` to align both programs, and the tactic `hyco_step` to match the two outputs. We note that in order to be able to prove that the event-invariant $\Box$ *double* is preserved after emitting the outputs, we critically need to use the memory-invariant (this corresponds to the instruction `rewrite <- INV` in the script). Once we proved that the event-invariant is maintained, we can conclude the proof with an application of `hyco_cycle`.

8 Related Work and Discussion

8.1 Game-Based Verification of Hyperliveness

Our approach is connected to a game-theoretic interpretation of hyperproperties introduced by Coenen et al. [18]. In their approach, the task of verifying a ∀∃ property is viewed as a game between a *universal player*, and an *existential player*. The moves of the universal player correspond to transitions in the left-hand system, and the existential player is forced to answer with corresponding transitions in the right-hand system. The existential player wins the game if there is a *strategy* to answer any move of the universal player without immediately violating the specification. In that case, the targeted hyperproperty holds. In the coalgebraic reading we presented in this paper, the greatest fixed point $\nu.\mathtt{feF}$ can be interpreted as the *winning region* of the game: the set of game states from which the existential player is guaranteed to win.

Originally, the game-based approach was introduced with the goal of automating the verification of ∀∃ temporal hyperproperties. The key insight is that when the systems being verified are finite-state, the game arena can be effectively constructed and sent to an efficient game solver. In the case of infinite-state systems described by programs, the arena cannot be constructed explicitly and we need to resort to finite approximations in order to automatically solve the game. For example, Beutner and Finkbeiner [12] used predicate abstraction to construct a finite approximation of the game.

An important advantage of the coalgebraic approach over the game-based approach is that it does not require us to explicitly construct an approximation of the the game. Instead, we implicitly define the *exact* wining region as a coinductive relation. This allows us to leverage parameterized coinduction as a deductive system to prove that the existential player has a winning strategy. Furthermore, by embedding our coalgebraic approach in a proof assistant such as Coq, we immediately benefit from its rich logic and its ecosystem of mathematical libraries to reason about the game.

8.2 Program Logics for Hyperproperties

There is a long tradition of verifying relational properties of sequential programs using extensions of Hoare logics (cf. [33]). *Relational Hoare Logic* (RHL) was originally introduced by Benton [9] and later extended to higher-order programs [1], separation logic [42], probabilistic computations [6], and quantum computations [7, 40]. D'Osualdo et al. introduced *hyper-triples* [22] as a unifying compositional building block for proofs of k-hypersafety properties. *Cartesian hoare logic* [38] is a sound and relatively complete calculus for k-safety properties.

Most relational logics are restricted to properties that express a condition over a given set of programs or a k-fold self-composition of some program for some fixed k. Some extensions that go beyond k-safety are directed at specific properties such as differential privacy [8] and sensitivity [5]. There are also several extensions that provide more general support for $\forall\exists$ hyperproperties. Dardinier and Müller [20] introduced *Hyper Hoare Logic*, a generalization of Hoare logic that lifts assertions to properties of arbitrary sets of states. Hyper Hoare Logic can reason about both the absence and the existence of combinations of executions. Other extensions to $\forall\exists$ hyperproperties include *Forall-Exist Hoare Tuples* (FEHT) [10], *refinement quadruples* [4] and *RHLE triples* [21]. Some of these approaches have been mechanized in a proof assistant (cf. [20]). However, all these approaches are limited to the analysis of pre-post style relational specifications. Unlike our approach, they are thus are not well-suited to reason about reactive systems whose executions are inherently infinite. By contrast, the coalgebraic approach goes beyond pre-post style specifications to reason about infinite sequences of events.

8.3 Mechanized Frameworks for Simulation Proofs

Interaction Trees (ITrees) [41] have recently been introduced as a general-purpose coinductive structure to model potentially non-terminating computations within the Coq proof assistant. Several approaches have been developed to establish trace inclusion (resp. equivalence) between ITrees using simulation (resp. bisimulation) techniques [15, 43]. Similar to the framework presented in this paper, these approaches are based on variants of parameterized coinduction. However, they do not support temporal reasoning. An interesting and natural direction would be to apply our approach to the verification of temporal hyperproperties of programs modeled as ITrees.

Another Coq-based framework for simulation proofs is SIMULIRIS [27]. It combines IRIS [30], a powerful concurrent separation logic, with simulation techniques. While SIMULIRIS does not target temporal hyperproperties, integrating our framework within a separation logic in a similar way is an interesting research direction. It would enable intuitive reasoning about temporal hyperproperties of heap-manipulating programs.

9 Conclusion

We have presented HYCO, a mechanized framework for the verification of temporal hyperproperties within the Coq proof assistant. Our approach provides a foundation for the construction of trustworthy proofs of temporal hyperproperties in complex reactive systems. Since we use the full logic of Coq as the underlying assertion language, the expressiveness of the hyperproperties considered here is far beyond the scope of currently available automated approaches. In particular, HYCO can easily be applied to new programming languages and new temporal logics.

In future work, we plan to build on our framework to design a fully-featured mechanized program logic for temporal hyperproperties of reactive systems. Another important direction is to investigate the completeness of the approach. It is well-known that the game-theoretic approach is incomplete in general [18]. For finite-state systems, the problem has been mitigated by adding *prophecy variables* that inform the existential player about future choices of the universal player [11]. Our proof system would likely benefit from the introduction of prophecy variables in a similar manner.

Data Availability

The Coq development accompanying this paper is available on GitHub at the following address:

https://github.com/acorrenson/hyco-popl-2025

Additionally, an archive containing the development version at the time this paper was submitted, together with detailed installation and evaluation instructions, can be found on Zenodo at the following address [19]:

https://zenodo.org/records/14055009

Acknowledgements

This work was supported by the European Research Council (ERC) Grant HYPER (No. 101055412). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. A. Correnson carried out this work as a member of the Saarbrücken Graduate School of Computer Science.

References

[1] Alejandro Aguirre, Gilles Barthe, Marco Gaboardi, Deepak Garg, and Pierre-Yves Strub. 2019. A relational logic for higher-order programs. *J. Funct. Program.* 29 (2019), e16. https://doi.org/10.1017/S0956796819000145

[2] Timos Antonopoulos, Paul Gazzillo, Michael Hicks, Eric Koskinen, Tachio Terauchi, and Shiyi Wei. 2017. Decomposition instead of self-composition for proving the absence of timing channels. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, Albert Cohen and Martin T. Vechev (Eds.). ACM, 362–375. https://doi.org/10.1145/3062341.3062378

[3] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking.* MIT Press, 249–253.

[4] Gilles Barthe, Juan Manuel Crespo, and César Kunz. 2013. Beyond 2-Safety: Asymmetric Product Programs for Relational Program Verification. In *Logical Foundations of Computer Science*, Sergei Artemov and Anil Nerode (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 29–43.

[5] Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2018. Proving expected sensitivity of probabilistic programs. *Proc. ACM Program. Lang.* 2, POPL (2018), 57:1–57:29. https://doi.org/10.1145/3158145

[6] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. 2009. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, Zhong Shao and Benjamin C. Pierce (Eds.). ACM, 90–101. https://doi.org/10.1145/1480881.1480894

[7] Gilles Barthe, Justin Hsu, Mingsheng Ying, Nengkun Yu, and Li Zhou. 2020. Relational proofs for quantum programs. *Proc. ACM Program. Lang.* 4, POPL (2020), 21:1–21:29. https://doi.org/10.1145/3371089

[8] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. 2013. Probabilistic Relational Reasoning for Differential Privacy. *ACM Trans. Program. Lang. Syst.* 35, 3 (2013), 9:1–9:49. https://doi.org/10.1145/2492061

[9] Nick Benton. 2004. Simple relational correctness proofs for static analyses and program transformations. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004, Venice, Italy, January 14-16, 2004*, Neil D. Jones and Xavier Leroy (Eds.). ACM, 14–25. https://doi.org/10.1145/964001.964003

[10] Raven Beutner. 2024. Automated Software Verification of Hyperliveness. In *Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 14571)*, Bernd Finkbeiner and Laura Kovács (Eds.). Springer, 196–216. https://doi.org/10.1007/978-3-031-57249-4_10

[11] R. Beutner and B. Finkbeiner. 2022. Prophecy Variables for Hyperproperty Verification. In *2022 2022 IEEE 35th Computer Security Foundations Symposium (CSF) (CSF).* IEEE Computer Society, Los Alamitos, CA, USA, 471–485. https://doi.org/10.1109/CSF54842.2022.9919658

[12] Raven Beutner and Bernd Finkbeiner. 2022. Software Verification of Hyperproperties Beyond k-Safety. In *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13371)*, Sharon Shoham and Yakir Vizel (Eds.). Springer, 341–362. https://doi.org/10.1007/978-3-031-13185-1_17

[13] Raven Beutner and Bernd Finkbeiner. 2023. AutoHyper: Explicit-State Model Checking for HyperLTL. In *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13993)*, Sriram Sankaranarayanan and Natasha Sharygina (Eds.). Springer, 145–163. https://doi.org/10.1007/978-3-031-30823-9_8

[14] Janusz A. Brzozowski. 1964. Derivatives of Regular Expressions. *J. ACM* 11, 4 (oct 1964), 481–494. https://doi.org/10.1145/321239.321249

[15] Minki Cho, Youngju Song, Dongjae Lee, Lennard Gäher, and Derek Dreyer. 2023. Stuttering for Free. *Proc. ACM Program. Lang.* 7, OOPSLA2, Article 281 (oct 2023), 28 pages. https://doi.org/10.1145/3622857

[16] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. 2014. Temporal Logics for Hyperproperties. In *Proc. POST*. 265–284.

[17] Michael R. Clarkson and Fred B. Schneider. 2008. Hyperproperties. In *2008 21st IEEE Computer Security Foundations Symposium*. 51–65. https://doi.org/10.1109/CSF.2008.7

[18] Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. 2019. Verifying Hyperliveness. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, 121–139. https://doi.org/10.1007/978-3-030-25540-4_7

[19] Arthur Correnson. 2024. *Coinductive Proofs for Temporal Hyperliveness*. https://doi.org/10.5281/zenodo.14055009

[20] Thibault Dardinier and Peter Müller. 2024. Hyper Hoare Logic: (Dis-)Proving Program Hyperproperties. *Proc. ACM Program. Lang.* 8, PLDI, Article 207 (jun 2024), 25 pages. https://doi.org/10.1145/3656437

[21] Robert Dickerson, Qianchuan Ye, Michael K. Zhang, and Benjamin Delaware. 2022. RHLE: Modular Deductive Verification of Relational ∀∃ Properties. In *Programming Languages and Systems: 20th Asian Symposium, APLAS 2022, Auckland, New Zealand, December 5, 2022, Proceedings* (Auckland, New Zealand). Springer-Verlag, Berlin, Heidelberg, 67–87. https://doi.org/10.1007/978-3-031-21037-2_4

[22] Emanuele D'Osualdo, Azadeh Farzan, and Derek Dreyer. 2022. Proving hypersafety compositionally. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 135 (oct 2022), 26 pages. https://doi.org/10.1145/3563298

[23] Javier Esparza, Peter Lammich, René Neumann, Tobias Nipkow, Alexander Schimpf, and Jan-Georg Smaus. 2013. A Fully Verified Executable LTL Model Checker. In *Computer Aided Verification*, Natasha Sharygina and Helmut Veith (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 463–478.

[24] Azadeh Farzan and Anthony Vandikas. 2019. Automated Hypersafety Verification. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, 200–218. https://doi.org/10.1007/978-3-030-25540-4_11

[25] Azadeh Farzan and Anthony Vandikas. 2020. Reductions for safety proofs. *Proc. ACM Program. Lang.* 4, POPL (2020), 13:1–13:28. https://doi.org/10.1145/3371081

[26] Bernd Finkbeiner, Markus N. Rabe, and Cesar Sanchez. 2015. Algorithms for Model Checking HyperLTL and HyperCTL*. In *Proc. CAV, 2015*.

[27] Lennard Gäher, Michael Sammler, Simon Spies, Ralf Jung, Hoang-Hai Dang, Robbert Krebbers, Jeehoon Kang, and Derek Dreyer. 2022. Simuliris: a separation logic framework for verifying concurrent program optimizations. *Proc. ACM Program. Lang.* 6, POPL, Article 28 (jan 2022), 31 pages. https://doi.org/10.1145/3498689

[28] Tzu-Han Hsu, César Sánchez, and Borzoo Bonakdarpour. 2021. Bounded Model Checking for Hyperproperties. In *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12651)*, Jan Friso Groote and Kim Guldstrand Larsen (Eds.). Springer, 94–112. https://doi.org/10.1007/978-3-030-72016-2_6

[29] 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* (Rome, Italy) *(POPL '13)*. Association for Computing Machinery, New York, NY, USA, 193–206. https://doi.org/10.1145/2429069.2429093

[30] Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Mumbai, India) *(POPL '15)*. Association for Computing Machinery, New York, NY, USA, 637–650. https://doi.org/10.1145/2676726.2676980

[31] Leslie Lamport and Fred B. Schneider. 2021. Verifying Hyperproperties With TLA. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*. IEEE, 1–16. https://doi.org/10.1109/CSF51468.2021.00012

[32] John McLean. 1994. A General Theory of Composition for Trace Sets Closed Under Selective Interleaving Functions. In *Proc. IEEE Symposium on Security and Privacy*. 79–93.

[33] David A. Naumann. 2020. Thirty-Seven Years of Relational Hoare Logic: Remarks on Its Principles and History. In *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer International Publishing, Cham, 93–116.

[34] Damien Pous. 2007. Complete Lattices and Up-To Techniques. In *Programming Languages and Systems*, Zhong Shao (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 351–366.

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

[36] Damien Pous and Davide Sangiorgi. 2011. *Enhancements of the bisimulation proof method.* Cambridge University Press, 233–289.

[37] Ron Shemer, Arie Gurfinkel, Sharon Shoham, and Yakir Vizel. 2019. Property Directed Self Composition. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11561)*, Isil Dillig and Serdar Tasiran (Eds.). Springer, 161–179. https://doi.org/10.1007/978-3-030-25540-4_9

[38] Marcelo Sousa and Isil Dillig. 2016. Cartesian hoare logic for verifying k-safety properties. *SIGPLAN Not.* 51, 6 (jun 2016), 57–69. https://doi.org/10.1145/2980983.2908092

[39] Peter Thiemann and Martin Sulzmann. 2015. From ω-Regular Expressions to Büchi Automata via Partial Derivatives. In *Language and Automata Theory and Applications*, Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe (Eds.). Springer International Publishing, Cham, 287–298.

[40] Dominique Unruh. 2019. Quantum relational Hoare logic. *Proc. ACM Program. Lang.* 3, POPL (2019), 33:1–33:31. https://doi.org/10.1145/3290346

[41] Li-yao Xia, Yannick Zakowski, Paul He, Chung-Kil Hur, Gregory Malecha, Benjamin C. Pierce, and Steve Zdancewic. 2019. Interaction trees: representing recursive and impure programs in Coq. *Proc. ACM Program. Lang.* 4, POPL, Article 51 (dec 2019), 32 pages. https://doi.org/10.1145/3371119

[42] Hongseok Yang. 2007. Relational separation logic. *Theor. Comput. Sci.* 375, 1-3 (2007), 308–334. https://doi.org/10.1016/j.tcs.2006.12.036

[43] 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* (New Orleans, LA, USA) *(CPP 2020)*. Association for Computing Machinery, New York, NY, USA, 71–84. https://doi.org/10.1145/3372885.3373813

Received 2024-07-11; accepted 2024-11-07