Rely-Guarantee Is Coinductive
– A Proof-Centered Investigation of Inductively Approximated Coinduction –

John Derrick, Chelsea Edmonds, Andrei Popescu, Jamie Wright

School of Computer Science, University of Sheffield

Abstract. We make the case that the foundation for Rely-Guarantee reasoning can be fruitfully delivered by a coinductive semantics. Using insight from an Isabelle formalization, via a proof analysis we show that the coinductive semantics tends to simplify the proof development; in particular it enables more direct proofs for the soundness of the Rely-Guarantee rules. The comparison between inductive and coinductive proofs also suggests inductive counterparts of coinductive "up-to" enhancements. On the way, we fill a gap in the literature, by showing that three previously defined inductive semantics for Rely-Guarantee are equivalent. Underlying our transformation of an inductive into a coinductive semantics is the notion of inductively approximating a coinductive predicate—which, deployed in the opposite direction (from coinduction to induction), is a standard technical tool for approximating process algebra bisimilarities. On the spectrum between the abstract fixpoint theorems and concrete instances, we formalize effective format-based criteria that enable sound approximation.

1 Introduction

The coinduction definition and proof method, which is the categorical dual of induction, has recently emerged as a powerful methodology for specifying and reasoning about systems. Roughly speaking, while induction is most suitable for describing finitary syntax and behavior, coinduction excels in the compact description of infinite system behavior [58,38,27,33,56,44,71].

The Rely-Guarantee method (§2) for the compositional verification of concurrent programs, introduced by Jones at the beginning of the 1980s [30,31], has had a major intellectual and practical influence on formal verification [25]. Several verification tools have incorporated ideas from Rely-Guarantee [48,18], whose application success has been further boosted by their combination with ideas from Separation Logic [50,57,28] which started with the work of Vafeiadis and Parkinson [67,65] on the "marriage" between the two logics.

Here we advocate another useful marriage, namely basing Rely-Guarantee on a coinductive foundation, instead of (or in addition to) its traditional inductive foundation (§3). While the first rigorous semantics for Rely-Guarantee, introduced by Xu et al. [72], employs a heavy inductive definition on interactive computation traces, in subsequent developments this account was simplified to use rely-directed reachability relations [13], and then further simplified to use counting-based

inductive safety predicates [67,65,66]—allowing for increasingly simpler and more transparent proofs of soundness for the Rely-Guarantee rules (§3.1).

Placing ourselves at the end of this simplification spectrum, we further push its boundaries: We show that an induction-to-coinduction shift in foundation can be performed by amending the counting-based semantics (§3.2). We argue that the coinductive semantics is the lightest and in some sense the most natural, and caters for the most direct (though admittedly less elementary) proofs of rule soundness. Furthermore, we formally prove all four semantics to be equivalent.

Then we engage in a proof-centered exploration of what makes the above shift possible: the sound inductive approximation of coinductive predicates, emerging from the properties of fixpoints in lattices via the Knaster-Tarski and Kleene theorems (§4). From our concrete analysis of inductive versus coinductive proofs of soundness for the Rely-Guarantee rules, we extract patterns of proof connections between inductive and coinductive predicates (§5). These allow us to explain the inductive-to-coinductive simplification in general terms, revealing the coinductive proof as the *core* of the inductive one (§5.1). Going deeper on the connection between coinduction and its approximating induction, we identify the inductive counterparts of coinductive "up-to" enhancements (§5.2).

Further exploring the scope of the inductive approximations of coinduction (§6), we formalize format-based criteria (§6.1, §6.3). These are easier to instantiate and to implement in program verifiers, and also cover the cases of (bi)similarity relations from process algebra (§6.2), and of Rely-Guarantee mixed with Separation Logic (§6.4). Capturing bisimilarities and Rely-Guarantee under a common format is particularly desirable, given that what we propose for Rely-Guarantee (switching from a well-established inductive to a coinductive notion) has been widely applied *in reverse* with bisimilarities, namely using induction to approximate a well-established coinductive notion.

Our findings have been driven by our experience of mechanizing in Isabelle the semantics of Rely-Guarantee while trying to simplify the "formalities" [15]. We take inspiration from, and create bridges between previous developments on Rely-Guarantee semantics, rule systems, and coinduction enhancements (§7).

More details and proofs of the results in this paper are provided in a technical report [16]. The report is identical to the paper, except for an appendix containing additional details and proofs.

Here is a summary of our novel contributions. On Rely-Guarantee, we give (1) the first coinductive semantics, and also (2) the first rigorous (and formal) proof of the equivalence of previously introduced inductive semantics. On inductive approximations for coinduction, we provide (3) a detailed comparison of inductive and coinductive proofs that starts with the Rely-Guarantee case study and extrapolates to general proof connections for least and greatest fixpoints of operators, (4) a novel inductive counterpart of coinductive up-to enhancements, shedding some "elementary" light on these enhancements, and (5) a formalization of general effective criteria that ensure ω-continuity (hence the soundness of the inductive approximations).

$$\frac{s' = \mathsf{evalA}\ a\ s}{(a, s) \Rightarrow (\mathsf{done}, s')}(\mathsf{Atom}) \qquad \frac{(c_1, s) \Rightarrow (c_1', s')}{(\mathsf{seq}\ c_1\ c_2, s) \Rightarrow (\mathsf{seq}\ c_1'\ c_2, s')}(\mathsf{Seq})$$

$$(\mathsf{seq}\ \mathsf{done}\ c, s) \Rightarrow (c, s) \quad (\mathsf{Seq\text{-}Done})$$

$$\frac{\mathsf{evalT}\ t\ s}{(\mathsf{if}\ t\ c_1\ c_2, s) \Rightarrow (c_1, s)}(\mathsf{If\text{-}True}) \qquad \frac{\neg\ \mathsf{evalT}\ t\ s}{(\mathsf{if}\ t\ c_1\ c_2, s) \Rightarrow (c_2, s)}(\mathsf{If\text{-}False})$$

$$(\mathsf{while}\ t\ c, s) \Rightarrow (\mathsf{if}\ t\ (\mathsf{seq}\ c\ (\mathsf{while}\ t\ c))\ \mathsf{done},\ s) \quad (\mathsf{While})$$

$$\frac{(c_1, s) \Rightarrow (c_1', s')}{(\mathsf{par}\ c_1\ c_2, s) \Rightarrow (\mathsf{par}\ c_1'\ c_2, s')}(\mathsf{Par\text{-}L}) \qquad \frac{(c_2, s) \Rightarrow (c_2', s')}{(\mathsf{par}\ c_1\ c_2, s) \Rightarrow (\mathsf{par}\ c_1\ c_2', s')}(\mathsf{Par\text{-}R})$$

$$(\mathsf{par}\ \mathsf{done}\ \mathsf{done}, s) \Rightarrow (\mathsf{done}, s) \quad (\mathsf{Par\text{-}Done})$$

Fig. 1: Small-step operational semantics

Notations and conventions. Bool denotes the two-element set of booleans, $\{\mathsf{true}, \mathsf{false}\}$. A predicate on a set A is a function of type $A \to \mathsf{Bool}$, and a relation between sets A and B is a function of type $A \to B \to \mathsf{Bool}$. For a predicate $P : A \to \mathsf{Bool}$ and $a \in A$, we write $P\ a$ instead of $P\ a = \mathsf{true}$, and read it as "P holds for a"; and similarly for relations. We use "RG" as an abbreviation for "Rely-Guarantee". We will sometimes use gray boxes to draw attention to certain parts of the formulas, often emphasizing changes or additions compared to previous formulas; these boxes do not have any mathematical meaning.

2 The Rely-Guarantee Reasoning Rules

We consider a simple imperative language with parallel composition whose syntax is given by the following grammar (where we have underlined the syntactic categories, Com, Atom and Test, in order to more clearly distinguish them from keywords/terminals such as done and while):

$$\begin{array}{rcl} \underline{\mathsf{Com}} & ::= & \mathsf{done} \mid \underline{\mathsf{Atom}} \mid \mathsf{seq}\ \underline{\mathsf{Com}}\ \underline{\mathsf{Com}} \mid \mathsf{if}\ \underline{\mathsf{Test}}\ \underline{\mathsf{Com}}\ \underline{\mathsf{Com}} \mid \\ & & \mathsf{while}\ \underline{\mathsf{Test}}\ \underline{\mathsf{Com}} \mid \mathsf{par}\ \underline{\mathsf{Com}}\ \underline{\mathsf{Com}} \end{array}$$

Thus, a command is either done (completed), an atom (atomic command), a sequential composition, an if-branching over a test, a while loop again regulated by a test, or the parallel composition of two commands. We leave the tests and atoms abstract. The small-step semantics, shown in Fig. 1, is quite standard. It indicates how command-state pairs, which we call *configurations*, evolve during a single execution step. Thus, the semantics is a (binary) relation $\Rightarrow$

between configurations. While the syntax is parameterized by the sets Test and Atom of tests and atoms respectively, the semantics is further parameterized by a set State of states, and by functions $\mathsf{evalT} : \mathsf{Test} \to \mathsf{State} \to \mathsf{Bool}$ and $\mathsf{evalA} : \mathsf{Atom} \to \mathsf{State} \to \mathsf{State}$ that evaluate tests and commands in any given state. Thus, given a state s and a test t, $\mathsf{evalT}\ t\ s$ returns the boolean value obtained by evaluating t in s. Similarly, given an atom a and two states s, s', $\mathsf{evalA}\ a\ s\ s'$ says that evaluating a starting in s can produce s'. The while loops proceed by unfolding into a conditional that nests a sequential composition, and parallel composition proceeds via shared-state concurrency.

A *Rely-Guarantee (RG) clause* is a tuple (P, R, G, Q) where $P : \mathsf{State} \to \mathsf{Bool}$ is the *pre-condition*, $R : \mathsf{State} \to \mathsf{State} \to \mathsf{Bool}$ the *rely-condition*, $G : \mathsf{State} \to \mathsf{State} \to \mathsf{Bool}$ the *guarantee-condition*, and $Q : \mathsf{State} \to \mathsf{Bool}$ the *post-condition*. We think of a rely-guarantee clause (P, R, G, Q) as making the following statement about a command c: If the execution of c starts in a state satisfying the pre-condition P, and if during the execution the environment is guaranteed to only change the state according to the rely-condition R, then the following hold: (1) each execution step of c will only change the state according to the guarantee-condition G, and (2) if the execution terminates, then the final state will satisfy the post-condition Q. This intuition will be made formal by the rely-guarantee semantics which we will discuss in §3.

Fig. 2 shows a standard RG proof system for reasoning compositionally about programs. It defines the relation $\vdash$ between commands and RG clauses inductively, i.e., as the smallest relation closed under the given rules. Besides the monotonicity rule (Mono), we have one rule for each language construct.

3 Four Ways of Making Sense of Rely-Guarantee

In this section we revisit three increasingly light inductive semantics for RG (§3.1), establish their equivalence, and propose our coinductive semantics further up on the lightness spectrum (§3.2).

A configuration (c, s) is *final*, written $\mathsf{final}\,(c, s)$, when there is no (c', s') such that $(c, s) \Rightarrow (c', s')$ (equivalently, when $c = \mathsf{done}$).

3.1 Three inductive ways

We next describe the three main semantics previously proposed for RG.

(i) The trace-based semantics of Xu, de Roever and He [72] was the first rigorous semantics proposed for RG. It uses *labeled configurations*, which are triples (l, c, s) such that (c, s) is a configuration and l is an element of the two-element set $\{\mathsf{C}, \mathsf{E}\}$, where C indicates a command step and E an environment step. An *(interactive computation) trace* is a nonempty list of labeled configurations $[(l^1, c^1, s^1), \ldots, (l^n, c^n, s^n)]$ such that, for all $i \in \{1, \ldots, n-1\}$, $l^{i+1} = \mathsf{C}$ implies $(c^i, s^i) \Rightarrow (c^{i+1}, s^{i+1})$, and $l^{i+1} = \mathsf{E}$ implies $c^{i+1} = c^i$. We write $\mathsf{Trace}(c)$ for the set of traces $[(l^1, c^1, s^1), \ldots, (l^n, c^n, s^n)]$ starting in c, i.e., such that $c^1 = c$.

$$\frac{c \vdash (P', R', G', Q') \qquad P \leq P' \qquad R \leq R' \qquad G' \leq G \qquad Q' \leq Q}{c \vdash (P, R, G, Q)}\,(\mathsf{Mono})$$

$$\frac{\mathsf{stable}\ Q\ R \qquad P \leq Q}{\mathsf{done} \vdash (P, R, G, Q)}\,(\mathsf{DoneRG})$$

$$\frac{\begin{array}{cc}\mathsf{stable}\ P\ R & \mathsf{stable}\ Q\ R \\ (\lambda s'.\ \exists s.\ P\ s \wedge \mathsf{evalA}\ a\ s\ s') \leq Q & (\lambda s, s'.\ P\ s \wedge \mathsf{evalA}\ a\ s\ s') \leq G\end{array}}{a \vdash (P, R, G, Q)}\,(\mathsf{AtomRG})$$

$$\frac{c_1 \vdash (P, R, G, P') \qquad c_2 \vdash (P', R, G, Q) \qquad \mathsf{refl}\ G}{\mathsf{seq}\ c_1\ c_2 \vdash (P, R, G, Q)}\,(\mathsf{SeqRG})$$

$$\frac{\begin{array}{cc}c_1 \vdash (P \sqcap (\mathsf{evalT}\ t), R, G, Q) & c_2 \vdash (P \sqcap (\neg\ (\mathsf{evalT}\ t)), R, G, Q) \\ \multicolumn{2}{c}{\mathsf{stable}\ P\ R \qquad \mathsf{refl}\ G}\end{array}}{\mathsf{if}\ t\ c_1\ c_2 \vdash (P, R, G, Q)}\,(\mathsf{IfRG})$$

$$\frac{\begin{array}{ccc}c \vdash (P \sqcap (\mathsf{evalT}\ t), R, G, P) & & P \sqcap (\neg\ (\mathsf{evalT}\ t)) \leq Q \\ \multicolumn{3}{c}{\mathsf{stable}\ P\ R \qquad \mathsf{stable}\ Q\ R \qquad \mathsf{refl}\ G}\end{array}}{\mathsf{while}\ t\ c \vdash (P, R, G, Q)}\,(\mathsf{WhileRG})$$

$$\frac{\begin{array}{c}c_1 \vdash (P_1, R_1, G_1, Q_1) \qquad c_2 \vdash (P_2, R_2, G_2, Q_2) \\ P \leq P_1 \sqcap P_2 \qquad R \sqcup G_2 \leq R_1 \qquad R \sqcup G_1 \leq R_2 \\ G_1 \sqcup G_2 \leq G \qquad Q_1 \sqcap Q_2 \leq Q \qquad \mathsf{refl}\ G\end{array}}{\mathsf{par}\ c_1\ c_2 \vdash (P, R, G, Q)}\,(\mathsf{ParRG})$$

Fig. 2: RG proof system. We write $\mathsf{refl}\ G$ to express that the relation G is reflexive. $\leq$ denotes the standard orders on predicates and relations, e.g., $P \leq P'$ means $\forall s.\ P\ s \longrightarrow P'\ s$, and similarly for relations. $\sqcap$ and $\sqcup$ denote the infimum and supremum in the lattices of predicates and relations, which are component-wise conjunction and disjunction. $\neg\ (\mathsf{evalT}\ t)$ denotes the componentwise negation of $\mathsf{evalT}\ t$, mapping each state s to $\neg\ (\mathsf{evalT}\ t\ s)$. Finally, $\mathsf{stable}\ P\ R$ says the predicate P is stable w.r.t. the relation R, i.e., $\forall s, s'.\ P\ s \wedge R\ s\ s' \longrightarrow P\ s'$.

The *Xu–de-Roever–He satisfaction* of an RG clause by a command, $c \vDash_{\mathsf{XRH}} (P, R, G, Q)$, is defined as follows: For all $tr = [(l^1, c^1, s^1), \ldots, (l^n, c^n, s^n)] \in \mathsf{Trace}(c)$, if $P\ s^1$ (the pre-condition holds at the beginning of tr) and $\forall i \in \{1, \ldots, n-1\}.\ l^{i+1} = \mathsf{E} \longrightarrow R\ s^i\ s^{i+1}$ (the rely-condition holds on all environment steps), then $\forall i \in \{1, \ldots, n-1\}.\ l^{i+1} = \mathsf{C} \longrightarrow G\ s^i\ s^{i+1}$ (the guarantee-condition holds on all computation steps) and $Q\ s^n$ (the post-condition holds at the end).

(ii) The reachability-based semantics of Coleman and Jones [13]. For a relation R on configurations, we define $\mathsf{stepRel}_R$ by the rules:

$$\frac{(c, s) \Rightarrow (c', s')}{\mathsf{stepRel}_R\ (c, s)\ (c', s')} \qquad \frac{R\ s\ s'}{\mathsf{stepRel}_R\ (c, s)\ (c, s')}$$

The reflexive-transitive closure of this relation, $\mathsf{stepRel}^*_R\ (c, s)\ (c', s')$ says that the configuration (c', s') is reachable from (c, s) through a combination of computation steps, and environment steps respecting the rely relation R. The *Coleman–Jones satisfaction* of an RG clause, $c \vDash_{\mathsf{CJ}} (P, R, G, Q)$, is defined as follows: For all s, c', s' such that $P\ s$ and $\mathsf{stepRel}^*_R\ (c, s)\ (c', s')$, we have that **(1)** $G\ s'\ s''$ for all c'', s'' with $(c', s') \Rightarrow (c'', s'')$, and **(2)** $\mathsf{final}\ (c', s')$ implies $Q\ s'$.

(iii) The counting-based (step-indexed) semantics of Vafeiadis and Parkinson [65,67]. This semantics uses a count of the number of interactive computation steps, and states that any number of such steps is "safe". While RG clauses are quadruples (P, R, G, Q), let us call any triple of the form (R, G, Q) a *reduced RG clause* (where the pre-condition was removed, i.e., retaining only the rely-, guarantee- and post-conditions). The notion of *safety* of a configuration (c, s) with respect to a reduced RG clause (R, G, Q) *within n execution steps*, $\mathsf{safe}_{(R,G,Q)}\ n\ (c, s)$, is defined inductively as shown in Fig. 3. Thus, every command in every state is safe after 0 execution steps, since of course nothing happened yet. For the inductive step, the predicate says that the $(n+1)$-safety of (c, s) (i.e., the safety of executing $n+1$ steps of c from state s) can be concluded from three hypotheses: **(1)** any rely-compliant environment step to s' (i.e., such that $R\ s\ s'$ holds) produces an n-safe configuration (c, s'); **(2)** the post-condition holds (i.e., $Q\ s$) in case (c, s) is final. **(3)** any computation step $(c, s) \Rightarrow (c', s')$ produces an n-safe configuration (c', s') along a guarantee-compliant state change (i.e., such that $G\ s\ s'$).

The *Vafeiadis–Parkinson satisfaction* of an RG clause, $c \vDash_{\mathsf{VP}} (P, R, G, Q)$, is now defined to mean safety for any number of steps, $\forall s \in \mathsf{State},\ \forall n \in \mathbb{N}.\ P\ s \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (c, s)$.

We can show that these three semantics are all equivalent, and the rules of the standard RG proof system are sound for them.

Thm 1. Consider the following statements:

(1) $c \vDash_{\mathsf{XRH}} (P, R, G, Q)$; (2) $c \vDash_{\mathsf{CJ}} (P, R, G, Q)$;
(3) $c \vDash_{\mathsf{VP}} (P, R, G, Q)$; (4) $c \vdash (P, R, G, Q)$.

Then (1), (2) and (3) are equivalent, and (4) implies them. Moreover, the equivalences between (1), (2) and (3) are language-independent, i.e., they still hold if we replace the small-step operational semantics of our language with any sets Com and State and relation $\Rightarrow : (\mathsf{Com} \times \mathsf{State}) \to (\mathsf{Com} \times \mathsf{State}) \to \mathsf{Bool}$.

$$\mathsf{safe}_{(R,G,Q)}\ 0\ (c, s)\ \ (\mathsf{Base})$$

$$\frac{\begin{array}{l}1.\ \forall s'.\ R\ s\ s' \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (c, s') \\ 2.\ \mathsf{final}\ (c, s)\ \longrightarrow Q\ s \\ 3.\ \forall c', s'.\ ((c, s) \Rightarrow (c', s')) \longrightarrow G\ s\ s' \wedge \mathsf{safe}_{(R,G,Q)}\ n\ (c', s')\end{array}}{\mathsf{safe}_{(R,G,Q)}\ (n+1)\ (c, s)}(\mathsf{Step})$$

Fig. 3: The inductive-safety predicate safe

Although equivalent, the three semantics differ in the heaviness of their inductive machinery:

- $\vDash_{\mathsf{XRH}}$ talks about long-distance action, i.e., quantifies over multiple execution steps, via explicit computation traces.
- $\vDash_{\mathsf{CJ}}$ still talks about long-distance action, but keeps the traces implicit, abstracted under the notion of reachability.
- $\vDash_{\mathsf{VP}}$ goes further, abstracting the (long-)distance into a numeric argument n, and only talks about single steps, n to $n+1$.

These differences are reflected in the level of formal bureaucracy involved in the proofs of RG rules soundness (the equivalence "(4) implies (i)" where $i \in \{1, 2, 3\}$ in Thm. 1). Take for example the soundness of the rule for sequential composition, (SeqRG) in Fig. 2:

- For $\vDash_{\mathsf{XRH}}$, where we need to prove that $c_1 \vDash_{\mathsf{XRH}} (P, R, G, P')$, $c_2 \vDash_{\mathsf{XRH}} (P', R, G, Q)$ and $\mathsf{refl}\ G$ implies $\mathsf{seq}\ c_1\ c_2 \vDash_{\mathsf{XRH}} (P, R, G, Q)$, we require a lemma characterizing traces that start in $\mathsf{seq}\ c_1\ c_2$, stating that such a trace:
  - either is obtained by wrapping $\mathsf{seq}\ \_\ c_2$ around a trace starting in c_1 (i.e., sequentially post-composing c_2 with all the commands of the trace),
  - or consists of a trace obtained by wrapping $\mathsf{seq}\ \_\ c_2$ around a trace starting in c_1 and ending in done, followed by a trace starting in c_2.
- For $\vDash_{\mathsf{CJ}}$ (where we must prove the same as above but with $\vDash_{\mathsf{CJ}}$), we require a lighter inversion lemma for multistep reachability: stating that, if $\mathsf{stepRel}^*_R(\mathsf{seq}\ c_1\ c_2, s)(c', s')$, then
  - either c' has the form $\mathsf{seq}\ c_1'\ c_2$ for some c_1' such that $\mathsf{stepRel}^*_R\ (c_1, s)\ (c_1', s')$,
  - or $\mathsf{stepRel}^*_R\ (c_1, s)\ (\mathsf{done}, s'')$ for some s'', and $\mathsf{stepRel}^*_R\ (c_2, s'')\ (c', s')$.
- Finally, for $\vDash_{\mathsf{VP}}$, we only require the native inversion lemma stemming from the inductive definition of the small-step semantics: stating that, if $(\mathsf{seq}\ c_1\ c_2, s) \Rightarrow (c', s')$, then
  - either c' has the form $\mathsf{seq}\ c_1'\ c_2$ for some c_1' such that $(c_1, s) \Rightarrow (c_1', s')$,
  - or $(c_1, s) \Rightarrow (\mathsf{done}, s'')$ for some s'', and $(c_2, s'') \Rightarrow (c', s')$.

So, proof bureaucracy decreases as we move along the above spectrum: from trace decomposition, to inversion for multi-steps, to inversion for single steps.

3.2 A fourth way: coinductive-safety semantics

Next, we discuss a further simplification from the lightest of the above semantics, the counting-based one of Vafeiadis and Parkinson. Taking advantage of the

$$\frac{\begin{array}{l}1.\ \forall s'.\ R\ s\ s' \longrightarrow \mathsf{safeC}_{(R,G,Q)}(c, s') \\ 2.\ \mathsf{final}\ (c, s) \longrightarrow Q\ s \\ 3.\ \forall c', s'.\ ((c, s) \Rightarrow (c', s')) \longrightarrow G\ s\ s' \wedge \mathsf{safeC}_{(R,G,Q)}\ (c', s')\end{array}}{\mathsf{safeC}_{(R,G,Q)}\ (c, s)}\ (\mathsf{StepC})$$

Fig. 4: The coinductive-safety predicate safeC

implicit (co)iterative nature of coinduction, we remove the explicit counting of the number n of steps from Fig. 3's inductive-safety predicate safe, and retain only the (Step) case. This leads us to the coinductive-safety predicate safeC defined by the rule schema in Fig. 4. The definition is now interpreted not inductively but coinductively [53,52,33], as indicated by a double-line in the rule. The *coinductive satisfaction* of an RG clause by a command, $c \vDash_{\mathsf{C}} (P, R, G, Q)$, is defined as $\forall s \in \mathsf{State}.\ P\ s \longrightarrow \mathsf{safeC}_{(R,G,Q)}\ (c, s)$ (so similarly to how the counting-based semantics $\vDash_{\mathsf{VP}}$ is defined from safe).

The coinductive semantics is equivalent to the counting-based one (hence, according to Thm. 1, to all three inductive semantics).

Thm 2. $c \vDash_{\mathsf{VP}} (P, R, G, Q)$ is equivalent to $c \vDash_{\mathsf{C}} (P, R, G, Q)$. Moreover, this equivalence is again language-independent (in the sense defined in Thm. 1).

The proof of the theorem relies on a lemma about safe versus safeC.

Lemma 3. The following are equivalent:
(i) $\forall n \in \mathbb{N}.\ \mathsf{safe}_{(R,G,Q)}\ n\ (c, s)$; (ii) $\mathsf{safeC}_{(R,G,Q)}\ (c, s)$.

The lemma says that a $\mathbb{N}$-indexed family of inductive predicates, $\mathsf{safe}_{(R,G,Q)}$, forms a convergent approximation of a coinductive predicate, $\mathsf{safeC}_{(R,G,Q)}$.

4 Coinduction Approximated Inductively in General

Next we recall the abstract phenomenon behind the inductive approximation of coinduction—as background for our proof-based investigation to follow.

Let $(L, \leq)$ be a complete lattice, where we write $\sqcap$ and $\sqcup$ for the binary infima and suprema, and $\bigsqcap$ and $\bigsqcup$ for infima and suprema of arbitrary families of elements in L. A *decreasing ω-chain* in L is a family $(l_i)_{i \in \mathbb{N}}$ such that $l_i \geq l_{i+1}$ for all $i \in \mathbb{N}$. Let $F : L \to L$ be a monotonic operator. An element $k \in L$ is said to be a *fixpoint* for F if $F\ k = k$, a *pre-fixpoint* for F if $F\ k \leq k$, and a *post-fixpoint* for F if $k \leq F\ k$. We recall the Knaster-Tarski theorem [63]:

Thm 4. If L is a complete lattice and $F : L \to L$ a monotonic operator, then: **(1)** There exists a unique least fixpoint lfp_F for F, which is also the least pre-fixpoint. **(2)** Dually, there exists a unique greatest fixpoint gfp_F for F, which is also the greatest post-fixpoint.

This enables an induction proof principle for lfp_F: to show $\mathsf{lfp}_F \leq k$, it suffices to show that k is a pre-fixpoint. Dually, we have a coinduction proof principle: to show $k \leq \mathsf{gfp}_F$, it suffices to show that k is a post-fixpoint. A small enhancement called *strong (co)induction* includes lfp_F and gfp_F too in the (co)inductive proofs:

Corollary 5. Under the hypotheses of Thm. 4:
(1) $F\,(\,\mathsf{lfp}_F \sqcap k) \leq k$ implies $\mathsf{lfp}_F \leq k$ for all $k \in L$, and
(2) $k \leq F\,(\,\mathsf{gfp}_F \sqcup k)$ implies $k \leq \mathsf{gfp}_F$ for all $k \in L$.

Example 6. The inductive definition of $\mathsf{safe}_{(R,G,Q)}$ comes from applying the Knaster-Tarski theorem for least fixpoints. Namely, for (R, G, Q) fixed, $\mathsf{safe}_{(R,G,Q)}$ is the least fixpoint of an operator H on the lattice $\mathbb{N} \to (\mathsf{Com} \times \mathsf{State}) \to \mathsf{Bool}$, defined, for each K, n and (c, s), by taking $H\ K\ n\ (c, s)$ to be

$$\begin{array}{l} n = 0 \ \vee \ (\exists m.\ n = m + 1 \wedge \\ \qquad\qquad (\mathsf{final}\ (c, s) \longrightarrow Q\ s) \wedge \\ \qquad\qquad (\forall c', s'.\ ((c, s) \Rightarrow (c', s')) \longrightarrow G\ s\ s' \wedge K\ (c', s')\ m) \wedge \\ \qquad\qquad (\forall s'.\ R\ s\ s' \longrightarrow K\ (c, s')\ m)) \end{array}$$

And the coinductive definition of $\mathsf{safeC}_{(R,G,Q)}$ comes from applying Knaster-Tarski for greatest fixpoints: $\mathsf{safeC}_{(R,G,Q)} = \mathsf{gfp}_F$ where the lattice L is $(\mathsf{Com} \times \mathsf{State}) \to \mathsf{Bool}$, and $F : L \to L$ is defined by taking $F\ K\ (c, s)$ to be

$$\begin{array}{l} (\mathsf{final}\ (c, s) \ \longrightarrow Q\ s) \wedge \\ (\forall c', s'.\ ((c, s) \Rightarrow (c', s')) \longrightarrow G\ s\ s' \wedge K\ (c', s')) \wedge \\ (\forall s'.\ R\ s\ s' \longrightarrow K\ (c, s')) \end{array}$$

We will denote by $\mathsf{L}_{(R,G,Q)}$ and $\mathsf{F}_{(R,G,Q)}$ the above particular lattice L and particular operator F underlying the coinductive definition of $\mathsf{safeC}_{(R,G,Q)}$. □

To capture abstractly the connection between safeC and safe we need a more fine-grained description of greatest fixpoints, via upper approximations. This is offered by Kleene's theorem [14], which we recall next (in the dual form, for greatest fixpoints). We say that $F : L \to L$ is *ω-cocontinuous* if it commutes with the infima of decreasing ω-chains $(a_i)_{i \in \mathbb{N}}$, i.e., $F\,(\bigsqcap_{i \in \mathbb{N}} a_i) = \bigsqcap_{i \in \mathbb{N}} (F\ a_i)$. Note that ω-cocontinuity is a strengthening of monotonicity. We write F^n for the n'th iteration of F, i.e., the composition with itself n times.

Thm 7. (Kleene's Theorem) If L is a complete lattice (or at least a copointed ω-cocomplete lattice) where $\top$ denotes its top element and the operator $F : L \to L$ is ω-cocontinuous, then $\mathsf{gfp}_F = \bigsqcap_{n \in \mathbb{N}} F^n \top$.

We recall the proof idea for future reference: The ω-cocontinuity of F ensures that $\bigsqcap_{n \in \mathbb{N}} F^n \top$ is a fixpoint. To show that it is the greatest, let k be a fixpoint. By induction on n it follows that $\forall n.\ k \leq F^n\ \top$, i.e., $k \leq \bigsqcap_{n \in \mathbb{N}} F^n\ \top$. In the inductive step, $k \leq F^n\ \top$ implies, also using F's monotonicity, that $k = F\ k \leq F\ (F^n\ \top) = F^{n+1}\ \top$. □

For any operator $F : L \to L$, we define its *approximating operator* $F^\sharp : (\mathbb{N} \to L) \to (\mathbb{N} \to L)$ as follows, for all $f : \mathbb{N} \to L$ and $n \in \mathbb{N}$:

$$F^\sharp\ f\ n = \begin{cases} \top, & \text{if } n = 0 \\ F\ (f\ (n-1)) & \text{otherwise} \end{cases}$$

The definition of $F^\sharp$ from F represents the pattern of how $\mathsf{safe}_{(R,G,Q)}$ *could have been* obtained from $\mathsf{safeC}_{(R,G,Q)}$. Indeed, by direct inspection of

the definitions, we can see that $\mathsf{safeC}_{(R,G,Q)}$ is $\mathsf{gfp}_{\mathsf{F}_{(R,G,Q)}}$ and $\mathsf{safe}_{(R,G,Q)}$ is $\mathsf{lfp}_{\mathsf{F}^\sharp_{(R,G,Q)}}$ (the least fixpoint of the approximating operator associated to $\mathsf{F}_{(R,G,Q)}$). Lemma 3, connecting $\mathsf{safe}_{(R,G,Q)}$ with $\mathsf{safeC}_{(R,G,Q)}$, is thus an instance of the following consequence of Kleene's theorem:

Thm 8. Assume that L is a complete lattice and $F : L \to L$ is an ω-cocontinuous operator. Then $\mathsf{gfp}_F = \bigsqcap_{n\in\mathbb{N}} \mathsf{lfp}_{F^\sharp} n$.

Thus, our discussed approximation phenomenon is abstractly the coincidence between the greatest fixpoint gfp_F and the infimum of the applications of the approximating operator's fixpoint $\mathsf{lfp}_{F^\sharp}\, n$; and this coincidence is enabled by the ω-cocontinuity of the underlying operator F.

5 Proof Development Perspective

The above discussion shows that, even though we derived $\mathsf{safeC}_{(R,G,Q)}$ from $\mathsf{safe}_{(R,G,Q)}$, it is more natural to see $\mathsf{safe}_{(R,G,Q)}$ as being derived from $\mathsf{safeC}_{(R,G,Q)}$, since $\mathsf{safeC}_{(R,G,Q)}$ is the simpler of the two. However, one is inductive and the other is coinductive, resulting in different styles of proofs, which we will compare next. We start with a concrete comparison, taking as a case study the soundness of the rule for sequential composition, then we extract general patterns in terms of the abstract operators $F^\sharp$ versus F (§5.1). Then we compare inductive with coinductive enhancements, starting with the soundness proofs of the RG rule for while (susceptible to "up-to" enhancement due to its reliance on other constructs), and again generalizing to abstract operators (§5.2).

5.1 Inductive versus coinductive proofs

Recall from §3.1 that the counting-based semantics is at the lightweight end of the spectrum of semantics for RG, with consequences on the simplicity of the soundness proofs. Next we illustrate how shifting from this to the coinductive semantics (i.e., from $\vDash_{\mathsf{VP}}$ which is based on safe to $\vDash_{\mathsf{C}}$ which is based on safeC), incurs a further degree of proof simplification. We again consider the rule for sequential composition ((SeqRG) in Fig. 2). The proofs of its soundness w.r.t. the counting-based and coinductive semantics are shown in Fig. 5, side by side. We arranged the layout and the notations to emphasize the similarity between their structures. Ignoring the specific formats required for starting (co)induction (i.e., the universal quantification for induction and the existential one for coinduction) and the vacuous base case present for induction only, the two proofs consist of very similar "face-offs" between what we know, labeled (3.i)–(3.iii), and what we must prove, labeled (i)–(iii). The differences, highlighted in gray in the figure, are:

- in the inductive proof, in addition to (3.i)–(3.iii) we also know (and of course need) the inductive hypothesis (IH);
- in the coinductive proof, in the goals (i) and (iii) we have the extra slack offered by a disjunct with the coinductive invariant (via Corollary 5(2)).

Preliminaries

We assume $\mathsf{refl}\ G$, $c_1 \vDash_{\mathsf{VP}} (P, R, G, P')$ and $c_2 \vDash_{\mathsf{VP}} (P', R, G, Q)$ and must show $\mathsf{seq}\ c_1\ c_2 \vDash_{\mathsf{VP}} (P, R, G, Q)$. Expanding the definition of $\vDash_{\mathsf{VP}}$ in the first assumption and the conclusion, the goal is reduced to the following, for all s and n:

Main part

We assume **(1)** $\mathsf{refl}\ G$ and **(2)** $\forall s', m.\ P'\ s' \longrightarrow \mathsf{safe}_{(R,G,Q)}\ m\ (c_2, s')$, and must prove $\forall c_1, s.\ (\forall m.\ \mathsf{safe}_{(R,G,P')}\ m\ (c_1, s)) \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{seq}\ c_1\ c_2, s)$, which we do by induction on n.

The base case is trivial, since the conclusion $\mathsf{safe}_{(R,G,Q)}\ 0$ is vacuously true.

For the induction step, we assume

(IH) $\forall c_1, s.\ (\forall m.\ \mathsf{safe}_{(R,G,P')}\ m\ (c_1, s)) \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{seq}\ c_1\ c_2, s)$

and must prove

$\forall c_1, s.\ (\forall m.\ \mathsf{safe}_{(R,G,P')}\ m\ (c_1, s)) \longrightarrow \mathsf{safe}_{(R,G,Q)}\ (n+1)\ (\mathsf{seq}\ c_1\ c_2, s)$.

To this end, we fix c_1 and s, and assume $\forall m.\ \mathsf{safe}_{(R,G,P')}\ m\ (c_1, s)$, in particular **(3)** $\forall m.\ \mathsf{safe}_{(R,G,P')}\ (m+1)\ (c_1, s)$, which by the definition of $\mathsf{safe}_{(R,G,P')}$ yields the following:

(3.i) $\forall m.\ \forall s'.\ R\ s\ s' \longrightarrow \mathsf{safe}_{(R,G,P')}\ m\ (c_1, s')$
(3.ii) $\mathsf{final}\ (c_1, s) \longrightarrow P'\ s$
(3.iii) $\forall m.\ \forall c_1', s'.\ ((c_1, s) \Rightarrow (c_1', s')) \longrightarrow G\ s\ s' \wedge \mathsf{safe}_{(R,G,P')}\ m\ (c_1', s')$

We must prove $\mathsf{safe}_{(R,G,Q)}\ (n+1)\ (\mathsf{seq}\ c_1\ c_2, s)$, which by the definition of $\mathsf{safe}_{(R,G,Q)}$ amounts to:

(i) $\forall s'.\ R\ s\ s' \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{seq}\ c_1\ c_2, s')$
(ii) $\mathsf{final}\ (\mathsf{seq}\ c_1\ c_2, s) \longrightarrow Q\ s$
(iii) $\forall c', s'.\ ((\mathsf{seq}\ c_1\ c_2, s) \Rightarrow (c', s')) \longrightarrow G\ s\ s' \wedge \mathsf{safe}_{(R,G,Q)}\ n\ (c', s')$

Now, (i) follows from (3.i) and (IH), and (ii) follows by virtue of $\mathsf{final}\ (\mathsf{seq}\ c_1\ c_2, s)$ being false. To prove (iii), fix c' and s' and assume $(\mathsf{seq}\ c_1\ c_2, s) \Rightarrow (c', s')$. We must prove $G\ s\ s' \wedge \mathsf{safe}_{(R,G,Q)}\ n\ (c', s')$. We have two cases:

Case I: $c_1 = \mathsf{done}$. Then $(c', s') = (c_2, s)$, hence $G\ s\ s'$, i.e., $G\ s\ s$, follows from (1). Moreover, since $\mathsf{final}\ (c_1, s)$ holds, from (3.ii) we obtain $P'\ s$, i.e., $P'\ s'$. This together with (2) implies $\mathsf{safe}_{(R,G,Q)}\ n\ (c_2, s')$, i.e., $\mathsf{safe}_{(R,G,Q)}\ n\ (c', s')$.

Case II: $c_1 \neq \mathsf{done}$. Then there exists c_1' such that $(c_1, s) \Rightarrow (c_1', s')$ and $c' = \mathsf{seq}\ c_1'\ c_2$, which together with (3.iii) implies $G\ s\ s'$ (as desired) and $\forall m.\ \mathsf{safe}_{(R,G,P')}\ m\ (c_1', s')$. Using the latter and (IH), we obtain $\mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{seq}\ c_1'\ c_2, s')$, i.e., $\mathsf{safe}_{(R,G,Q)}\ n\ (c', s')\ (R, G, Q)$. □

(a) For the counting-based semantics

Preliminaries

We assume $\mathsf{refl}\ G$, $c_1 \vDash_{\mathsf{C}} (P, R, G, P')$ and $c_2 \vDash_{\mathsf{C}} (P', R, G, Q)$ and must show $\mathsf{seq}\ c_1\ c_2 \vDash_{\mathsf{C}} (P, R, G, Q)$. Expanding the definition of $\vDash_{\mathsf{C}}$ in the first assumption and the conclusion, the goal is reduced to the following, for all s:

Main part

We assume **(1)** $\mathsf{refl}\ G$ and **(2)** $\forall s'.\ P'\ s' \longrightarrow \mathsf{safeC}_{(R,G,Q)}\ (c_2, s')$, and must prove $(\exists c_1.\ c = \mathsf{seq}\ c_1\ c_2 \wedge \mathsf{safeC}_{(R,G,P')}(c_1, s)) \longrightarrow \mathsf{safeC}_{(R,G,Q)}\ (c, s)$. We prove this by strong coinduction on the definition of $\mathsf{safeC}_{(R,G,Q)}$. Thus, we assume the coinductive invariant $\exists c_1.\ c = \mathsf{seq}\ c_1\ c_2 \wedge \mathsf{safeC}_{(R,G,P')}(c_1, s)$, which amounts to fixing c_1 and assuming $c = \mathsf{seq}\ c_1\ c_2$ and **(3)** $\mathsf{safeC}_{(R,G,P')}\ (c_1, s)$, the latter yielding the following from the definition of $\mathsf{safeC}_{(R,G,P')}$:

(3.i) $\forall s'.\ R\ s\ s' \longrightarrow \mathsf{safeC}_{(R,G,P')}\ (c_1, s')$
(3.ii) $\mathsf{final}\ (c_1, s) \longrightarrow P'\ s$
(3.iii) $\forall c_1', s'.\ ((c_1, s) \Rightarrow (c_1', s')) \longrightarrow G\ s\ s' \wedge \mathsf{safeC}_{(R,G,P')}\ (c_1', s')$

We must show the following:

(i) $\forall s'.\ R\ s\ s' \longrightarrow \mathsf{safeC}_{(R,G,Q)}\ (\mathsf{seq}\ c_1\ c_2, s') \vee (\exists c_1'.\ \mathsf{seq}\ c_1\ c_2 = \mathsf{seq}\ c_1'\ c_2 \wedge \mathsf{safeC}_{(R,G,P')}(c_1', s))$
(ii) $\mathsf{final}\ (\mathsf{seq}\ c_1\ c_2, s) \longrightarrow Q\ s$
(iii) $\forall c', s'.\ ((\mathsf{seq}\ c_1\ c_2, s) \Rightarrow (c', s')) \longrightarrow G\ s\ s' \wedge (\mathsf{safeC}_{(R,G,Q)}\ (c', s') \vee (\exists c_1'.\ c' = \mathsf{seq}\ c_1'\ c_2 \wedge \mathsf{safeC}_{(R,G,P')}(c_1', s')))$

Now, (i) follows from (3.i) taking $c_1' = c_1$, and (ii) follows by virtue of $\mathsf{final}\ (\mathsf{seq}\ c_1\ c_2, s)$ being false. To prove (iii), fix c' and s' and assume $(\mathsf{seq}\ c_1\ c_2, s) \Rightarrow (c', s')$. We must prove $G\ s\ s' \wedge$ (*) where (*) is $\mathsf{safeC}_{(R,G,Q)}\ (c', s')) \vee (\exists c_1'.\ c' = \mathsf{seq}\ c_1'\ c_2 \wedge \mathsf{safeC}_{(R,G,P')}(c_1', s'))$. We have two cases:

Case I: $c_1 = \mathsf{done}$. Then $(c', s') = (c_2, s)$, hence $G\ s\ s'$, i.e., $G\ s\ s$, follows from (1). Moreover, since $\mathsf{final}\ (c_1, s)$ holds, from (3.ii) we obtain $P'\ s$, i.e., $P'\ s'$. This together with (2) implies $\mathsf{safeC}_{(R,G,Q)}\ (c_2, s')$, i.e., $\mathsf{safeC}_{(R,G,Q)}\ (c', s')$, the first disjunct of (*).

Case II: $c_1 \neq \mathsf{done}$. Then there exists c_1' such that $(c_1, s) \Rightarrow (c_1', s')$ and $c' = \mathsf{seq}\ c_1'\ c_2$, which with (3.iii) implies $G\ s\ s'$ (as desired) and $\mathsf{safeC}_{(R,G,P')}\ (c_1', s')$. Using the latter together with $c' = \mathsf{seq}\ c_1'\ c_2$, we obtain the second disjunct of (*). □

(b) For the coinductive semantics

Fig. 5: Soundness proofs for the RG seq rule

The coinductive proof is more direct. Indeed, for inferring (i)–(iii) from (3.i)–(3.iii), the inductive version involves a back-and-forth between the premise and the conclusion of the fact to be proved, mediated by the inductive hypothesis. For example, when proving (iii) while in Case II, the inductive argument obtains $\forall m.\ \mathsf{safe}_{(R,G,P')}\, m\, (c_1', s')$ which is fed as premise of the inductive hypothesis to produce the desired $\mathsf{safe}_{(R,G,Q)}\, n\, (\mathsf{seq}\ \ c_1'\ \ c_2, s')$. (Note that the premise cannot be decoupled from the fact to be proved inductively, since the state in the premise can change during proof.) By contrast, in the coinductive version one obtains the corresponding fact $\mathsf{safeC}_{(R,G,P')}(c_1', s')$, fed directly to (iii) via the coinductive invariant component.

This conceptual simplification offered by the coinductive semantics can be traced back to the abstract operators F and $F^\sharp$, generalizing $\mathsf{safeC}_{(R,G,Q)}$ and $\mathsf{safe}_{(R,G,Q)}$ to gfp_F and $\mathsf{lfp}_{F^\sharp}$. The nontrivial part of an inductive proof of $k \le \mathsf{gfp}_F = (\bigsqcap_{n\in\mathbb{N}} \mathsf{lfp}_{F^\sharp}\, n)$, i.e., of $\forall n \in \mathbb{N}.\ k \le \mathsf{lfp}_{F^\sharp}\, n$, is to show that $k \le \mathsf{lfp}_{F^\sharp}\, n$ implies $k \le \mathsf{lfp}_{F^\sharp}(n+1)$. And oftentimes, in particular for the soundness proofs of all RG rules (including the Fig. 5 one), the reason why this implication holds is that $k \le F\,(\mathsf{gfp}_F \sqcup k)$—exactly what a proof of $k \le \mathsf{gfp}_F$ by strong coinduction entails. Indeed, we infer $k \le \mathsf{lfp}_{F^\sharp}(n+1)$ from $k \le \mathsf{lfp}_{F^\sharp}\, n$ by plugging $k \le F\,(\mathsf{gfp}_F \sqcup k)$ into the following chain of (in)equalities:

$$k \le F\,(\mathsf{gfp}_F \sqcup k) \le F\,(\mathsf{gfp}_F \sqcup \mathsf{lfp}_{F^\sharp}\, n) = F\,(\mathsf{lfp}_{F^\sharp}\, n) = F^\sharp\, \mathsf{lfp}_{F^\sharp}\, (n{+}1) = \mathsf{lfp}_{F^\sharp}(n{+}1)$$

(where, from left to right: the second inequality and the first equality follow from $k \le \mathsf{lfp}_{F^\sharp}(n)$, $\mathsf{gfp}_F \le \mathsf{lfp}_{F^\sharp}(n)$ and the monotonicity of F; the second equality follows from the definition of $F^\sharp$; and the third equality follows form $\mathsf{lfp}_{F^\sharp}$ being a fixpoint of $F^\sharp$). Thus, the coinductive proof corresponds to the core of the inductive proof; and the rest of the proof is boilerplate, essentially repeating the argument for Kleene's theorem (Thm. 7)!

So in our concrete example from Fig. 5, gfp_F is $\mathsf{safeC}_{(R,G,Q)}$, $\mathsf{lfp}_{F^\sharp}$ is $\mathsf{safe}_{(R,G,Q)}$ (so F is $\mathsf{F}_{(R,G,Q)}$), and k is expressed by the coinductive invariant. To see more clearly that our abstract discussion actually matches the concrete Fig. 5 situation, we concretize one notch of the abstract discussion, assuming L is $A \to \mathsf{Bool}$ for some set A and $k : A \to \mathsf{Bool}$ is existentially quantified over parameters from a set Prm, i.e., $k\, a = (\exists p.\ k'\, p\, a)$ for some $k' : Prm \to A \to \mathsf{Bool}$. Then:

- The inductive version proves $k \le \mathsf{gfp}_F = \bigsqcap_n \mathsf{lfp}_{F^\sharp}\, n$, i.e., $\forall n, a.\ k\ a \longrightarrow \mathsf{lfp}_{F^\sharp}\, n\ a$, i.e., $\forall n, a, p.\ k'\ p\ a \longrightarrow \mathsf{lfp}_{F^\sharp}\, n\ a$. Hence, the inductive goal is the following: $\forall a, p.\ k'\ p\ a \longrightarrow \mathsf{lfp}_{F^\sharp}\, n\ a$ implies $\forall a, p.\ k'\ p\ a \longrightarrow \mathsf{lfp}_{F^\sharp}(n+1)\ a$.
- The coinductive version proves $k \le \mathsf{gfp}_F$, i.e., $\forall a.\ k\ a \longrightarrow \mathsf{gfp}_F\ a$, i.e., $\forall a.\ (\exists p.\ k'\ p\ a) \longrightarrow \mathsf{gfp}_F\ a$. Hence, coinductively, the goal is the following: $\exists p.\ k'\ p\ a$ implies $F\,(\mathsf{gfp}_F \sqcup (\exists p.\ k'\ p))\ a$.

In our example from Fig. 5, A is $\mathsf{Com} \times \mathsf{State}$, Prm is Com, and k' is defined by $k'\ c_1\ (c, s) \longleftrightarrow (c = \mathsf{seq}\ c_1\ c_2 \wedge \mathsf{safeC}_{(R,G,P')}(c_1, s))$.

5.2 Inductive versus coinductive enhancements

As seen in Fig. 5, the soundness of the RG rule for seq w.r.t. the inductive (counting-based) and coinductive semantics amounts to seq preserving the

	Safety preservation needed for inductive soundness	Enhanced version
done	$\dfrac{(1)\ Q\ s \qquad (2)\ \mathsf{stable}\ Q\ R}{\forall n.\ \mathsf{safe}_{(R,G,Q)}\ n\ \mathsf{done}}$	None needed, this is just rephrasing
seq	$\dfrac{(1)\ \mathsf{refl}\ G \quad (2)\ \forall m\ .\ \mathsf{safe}_{(R,G,P)}\ m\ (c_1, s) \quad (3)\ \forall s'.\ P\ s' \longrightarrow \forall n.\ \mathsf{safe}_{(R,G,Q)}\ n\ (c_2, s')}{\forall n.\ \mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{seq}\ c_1\ c_2, s)}$	for all n ... $\dfrac{(1)\ \mathsf{refl}\ G \quad (2)\ \mathsf{safe}_{(R,G,P)}\ n\ (c_1, s) \quad (3)\ \forall s'.\ P\ s' \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (c_2, s')}{\mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{seq}\ c_1\ c_2, s)}$
if	$\dfrac{(1)\ \mathsf{refl}\ G \quad (2)\ \forall s'.\ R^*\ s\ s' \longrightarrow (\mathsf{evalT}\ t\ s' \longrightarrow \forall n.\ \mathsf{safe}_{(R,G,Q)}\ n\ (c_1, s')) \wedge (\neg\ \mathsf{evalT}\ t\ s' \longrightarrow \forall n.\ \mathsf{safe}_{(R,G,Q)}\ n\ (c_2, s'))}{\forall n.\ \mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{seq}\ c_1\ c_2, s)}$	for all n ... $\dfrac{(1)\ \mathsf{refl}\ G \quad (2)\ \forall s'.\ R^*\ s\ s' \longrightarrow (\mathsf{evalT}\ t\ s' \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (c_1, s')) \wedge (\neg\ \mathsf{evalT}\ t\ s' \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (c_2, s'))}{\mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{seq}\ c_1\ c_2, s)}$

Fig. 6: Inductive enhancements

safety predicates. The same is true for the other language constructs. For example, Fig. 6(left) shows the safety preservation properties needed for the soundness of the RG rules for done, seq and if w.r.t. the inductive semantics.

Now let us look at the soundness proofs for the RG rule of the while construct ((WhileRG) from Fig. 2), which is interesting because its operational semantics relies on other constructs, namely done, seq and if. Fig. 9 shows the soundness proofs relative to the two semantics, side by side. In the inductive case, we end up having to prove the implication shown in Fig. 9(a)'s box, which amounts to $\mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{while}\ t\ c, s)$ implies $\mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{if}\ t\ (\mathsf{seq}\ c\ (\mathsf{while}\ t\ c))\ \mathsf{done}, s)$. Here, safety preservation properties for if, seq and done would of course be handy. But the ones from Fig. 6(left) for if and seq, which *were* sufficient for proving the inductive soundness of the rules for these operators, are not strong enough for helping out with the proof for while. This is because now we need to work index-wise with a fixed n. Fortunately, the index-wise versions of these properties, shown in Fig. 6(right), are also provable—for example, we can prove that safety holds for $\mathsf{seq}\ c_1\ c_2$ in the same number of steps n as the assumed safety for c_2. And indeed, as shown in Fig. 9(a) after the box, these stronger, index-wise versions are just what we need to complete the proof for while. In summary, since while depends on other operators, its safety preservation property needed for the soundness of its RG rule required an index-wise enhancement of the safety preservation properties needed for the soundness of these operators. But what is the corresponding phenomenon within the coinductive semantics?

Before answering this question, let us first rephrase and then generalize the preservation properties for inductive safety. All the original preservation

$$
\begin{aligned}
\mathsf{U}^{\mathsf{done}}_{Q}\,(c,s) &= c = \mathsf{done} \wedge Q\,s \\
\mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}\,K\,(c,s) &= \exists c_1.\ c = \mathsf{seq}\ c_1\ c_2 \wedge \mathsf{safe}_{(R,G,P)}(c_1,s) \wedge \\
&\qquad (\forall s'.\ P\ s' \longrightarrow K\,(c_2,s')) \\
\mathsf{U}^{\mathsf{if}}_{(R,t,c_1,c_2)}\,(K_1,K_2)\,(c,s) &= (c = \mathsf{if}\ t\ c_1\ c_2 \wedge \\
&\qquad \forall s'.\ R^*\ s\ s' \longrightarrow (\mathsf{evalT}\ t\ s' \longrightarrow K_1\,(c_1,s')) \wedge \\
&\qquad\qquad (\neg\,\mathsf{evalT}\ t\ s' \longrightarrow K_2\,(c_2,s')))
\end{aligned}
$$

Fig. 7: Operators underlying done, seq and if (more details in [16, App. D])

properties from Fig. 6(left) can be stated (under side conditions) as

$$U\,(\bigsqcap_n \mathsf{safe}_{R,G,Q}\,n,\ldots,\bigsqcap_n \mathsf{safe}_{R,G,Q}\,n) \le \bigsqcap_n \mathsf{safe}_{R,G,Q}\,n$$

where U is an m-ary operator on the lattice $\mathsf{L}_{(R,G,Q)}$ (in our case with $m \in \{0,1,2\}$). Similarly, their index-wise enhancements from Fig. 6(right) can be stated (under the same conditions) as

$$\forall n.\, U(\mathsf{safe}_{R,G,Q}\,n,\ldots,\mathsf{safe}_{R,G,Q}\,n) \le \mathsf{safe}_{R,G,Q}\,n\,.$$

For example, the operators for done, seq, if are $\mathsf{U}^{\mathsf{done}}_{Q}$, $\mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}$, $\mathsf{U}^{\mathsf{if}}_{(R,t,c_1,c_2)}$, shown in Fig. 7; they were extracted from the soundness preservation properties.

So in general, for an ω-cocomplete operator $F : L \to L$ on a complete lattice L, where we have $\mathsf{gfp}_F = \bigsqcap_n \mathsf{lfp}_{F^\sharp}\,n$, a gfp_F-preservation property has the form

$$(*) \qquad U\,(\mathsf{gfp}_F,\ldots,\mathsf{gfp}_F) \le \mathsf{gfp}_F$$

where $U : L^m \to L$; and an inductive enhancement of it has the form

$$(**) \qquad \forall n.\, U\,(\mathsf{lfp}_{F^\sharp}\,n,\ldots,\mathsf{lfp}_{F^\sharp}\,n) \le \mathsf{lfp}_{F^\sharp}\,n\,.$$

And indeed, (**) is stronger than (*) if U itself is ω-cocontinuous (which *is* the case with our safety preservation operators).

Next, we will infer the coinductive version of enhancement by analyzing the typical pattern of a proof of the inductive enhancement (**) by induction on n, and (similarly to what we did in §5.1) trying to identify its coinductive core. For the inductive step, assuming $U\,(\mathsf{lfp}_{F^\sharp}\,n,\ldots,\mathsf{lfp}_{F^\sharp}\,n) \le \mathsf{lfp}_{F^\sharp}\,n$, one typically proves $U\,(\mathsf{lfp}_{F^\sharp}\,(n+1),\ldots,\mathsf{lfp}_{F^\sharp}\,(n+1)) \le \mathsf{lfp}_{F^\sharp}\,(n+1)$ via reasoning that amounts to the following chain of (in)equalities:

$$
\begin{aligned}
&U(\mathsf{lfp}_{F^\sharp}(n+1),\ldots,\mathsf{lfp}_{F^\sharp}\,(n+1)) = U(\ F(\mathsf{lfp}_{F^\sharp}n)\ ,\ldots,\ F(\mathsf{lfp}_{F^\sharp}n)\ ) \\
&\le F(\ \mathsf{lfp}_{F^\sharp}n \sqcup U(\ \mathsf{lfp}_{F^\sharp}n\ ,\ldots,\ \mathsf{lfp}_{F^\sharp}n\ )) = F(\mathsf{lfp}_{F^\sharp}n) = \mathsf{lfp}_{F^\sharp}(n+1)
\end{aligned}
$$

In concrete cases, the highlighted inequality typically holds regardless of whether we have $F\,(\mathsf{lfp}_{F^\sharp}\,n)$ on the left and $\mathsf{lfp}_{F^\sharp}\,n$ on the right, except for the fact that $F\,(\mathsf{lfp}_{F^\sharp}\,n) \le \mathsf{lfp}_{F^\sharp}\,n$; in other words, we could equally well prove that

$$(***) \qquad U\,(k_1,\ldots,k_m) \le F\,((k_1' \sqcup \ldots \sqcup k_m') \sqcup U\,(k_1,\ldots,k_m))$$

whenever $k_i \le k_i'$ and $k_i \le F\,k_i'$ for all $i \in \{1,\ldots,m\}$.

Thus, at the core of an inductive enhancement proof, we discovered property (***), which happens to be a variation of a well-known concept. Let us

	Safety preservation needed for coinductive soundness	Enhanced version
done	$\dfrac{(1)\ Q\ s \qquad (2)\ \mathsf{stable}\ Q\ R}{\mathsf{safeC}_{(R,G,Q)}\ (\mathsf{done}, s)}$ i.e., $\dfrac{(1)\ Q\ s \quad (2)\ \mathsf{stable}\ Q\ R}{\mathsf{U}^{\mathsf{done}}_{Q} \le \mathsf{safeC}_{(R,G,Q)}}$ proved by strong coinduction in the form: $\dfrac{(1)\ Q\ s \qquad (2)\ \mathsf{stable}\ Q\ R}{\mathsf{U}^{\mathsf{done}}_{Q}\ (c, s) \le \mathsf{F}_{(R,G,Q)}\ (\mathsf{safe}_{(R,G,Q)} \sqcup \mathsf{U}^{\mathsf{done}}_{Q})}$	No enhancement needed for 0-ary operators such as $\mathsf{U}^{\mathsf{done}}_{Q}$.
seq	$\dfrac{(1)\ \mathsf{refl}\ G \qquad (2)\ \mathsf{safeC}_{(R,G,P)}(c_1, s)}{\mathsf{safeC}_{(R,G,Q)}(\mathsf{seq}\ c_1\ c_2, s)}$ i.e., $\dfrac{(1)\ \mathsf{refl}\ G}{\mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}\mathsf{safeC}_{(R,G,Q)} \le \mathsf{safeC}_{(R,G,Q)}}$ proved by strong coinduction in the form: $\dfrac{(1)\ \mathsf{refl}\ G}{\mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}\ \mathsf{safeC}_{(R,G,Q)}\ \le \mathsf{F}_{(R,G,Q)}(\ \mathsf{safe}_{(R,G,Q)}\ \sqcup\ \mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}\ \mathsf{safeC}_{(R,G,Q)}\ )}$	$\dfrac{(1)\ \mathsf{refl}\ G}{K \le K' \qquad K \le \mathsf{F}_{(R,G,Q)}\ K'}$ $\mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}\ K\ \le$ $\mathsf{F}_{(R,G,Q)}(\ K'\ \sqcup \mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}\ K')$
if	$\dfrac{\begin{array}{c}(1)\ \mathsf{refl}\ G \\ (2)\ \forall s'.\ R^*\ s\ s' \longrightarrow (\mathsf{evalT}\ t\ s' \longrightarrow \mathsf{safeC}_{(R,G,Q)}(c_1, s')) \wedge \\ (\neg\ \mathsf{evalT}\ t\ s' \longrightarrow \mathsf{safeC}_{(R,G,Q)}(c_2, s'))\end{array}}{\mathsf{safeC}_{(R,G,Q)}(\mathsf{if}\ t\ c_1\ c_2, s)}$ i.e., $\dfrac{(1)\ \mathsf{refl}\ G}{\mathsf{U}^{\mathsf{if}}_{(R,t,c_1,c_2)}(\mathsf{safeC}_{(R,G,Q)}, \mathsf{safeC}_{(R,G,Q)}) \le \mathsf{safeC}_{(R,G,Q)}}$ proved by strong coinduction in the form: $\dfrac{(1)\ \mathsf{refl}\ G}{\begin{array}{l}\mathsf{U}^{\mathsf{if}}_{(R,t,c_1,c_2)}(\ \mathsf{safeC}_{(R,G,Q)}\ ,\ \mathsf{safeC}_{(R,G,Q)}\ )\ \le \\ \mathsf{F}_{(R,G,Q)}(\ \mathsf{safe}_{(R,G,Q)}\ \sqcup \\ \qquad \mathsf{U}^{\mathsf{if}}_{(R,t,c_1,c_2)}(\ \mathsf{safeC}_{(R,G,Q)}\ ,\ \mathsf{safeC}_{(R,G,Q)}\ ))\end{array}}$	$\dfrac{(1)\ \mathsf{refl}\ G}{\begin{array}{l}K_1 \le K_1' \qquad K_1 \le \mathsf{F}_{(R,G,Q)}\ K_1' \\ K_2 \le K_2' \qquad K_2 \le \mathsf{F}_{(R,G,Q)}\ K_2'\end{array}}$ $\mathsf{U}^{\mathsf{if}}_{(R,t,c_1,c_2)}(\ K_1\ ,\ K_2\ ) \le$ $\mathsf{F}_{(R,G,Q)}(\ K_1'\ \sqcup\ K_2'\ \sqcup$ $\mathsf{U}^{\mathsf{if}}_{(R,t,c_1,c_2)}(\ K_1'\ ,\ K_2'\ ))$

Fig. 8: Coinductive enhancements

call an operator $U : L^m \to L$ *weakly F-respectful* if it satisfies (***); and *F-respectful* if it satisfies (***) even without the $\sqcup$ part, i.e., with conclusion $U\,(k_1, \ldots, k_m) \leq F\,(U\,(k'_1, \ldots, k'_m))$. Unary respectful operators were studied by Sangiorgi and Pous [59,54], who proved that they enable *up-to enhancements* of coinduction. This generalizes to weakly respectful m-ary operators:

Thm 9. If $F : L \to L$ is a monotonic operator on a complete lattice and $U : L^m \to L$ is monotonic and weakly F-respectful, then the following *up-to-U coinduction* principle holds: Given $k_1, \ldots, k_m \in L$, for proving $k_i \leq \mathsf{gfp}_F$ for all $i \in \{1, \ldots, m\}$, it suffices to prove $k_i \leq F\,(U(k_1, \ldots, k_m))$ for all $i \in \{1, \ldots, m\}$.

Assuming $m = 1$, the principle allows inferring $k \leq \mathsf{gfp}_F$ from $k \leq F(U\,k)$, which generalizes the Knaster-Tarski coinduction principle: taking $U = 1_L$ we obtain ordinary coinduction, and taking $U = \lambda k.\ \mathsf{gfp}_F \sqcup k$ we obtain strong coinduction; overall, it offers more flexibility in a conductive proof, since $U\,k$ is usually larger than k. The class of (weakly) respectful operators contains many useful basic operators such as the projections. It is closed under composition, in that $V \circ (U_1, \ldots, U_n)$ is (weakly) respectful whenever V and $U_1, \ldots, U_n$ are, and suprema, in that $U_1 \sqcup U_2$ is (weakly) respectful whenever U_1 and U_2 are. Thus, we can obtain coinductive enhancements by quite freely combining weakly respectful operators.

Back to our concrete situation, Fig. 8(right) shows the conditions under which the operators $\mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}$ and $\mathsf{U}^{\mathsf{if}}_{(R,t,c_1,c_2)}$ are weakly respectful. Indeed, the (conditional) weak respectfulness of these operators represent the coinductive enhancements we are after. Moreover, the gray highlighting in Fig. 8 shows how the weak respectfulness properties (from the right) can be regarded as a generalization of the coinductive goal arising when proving the corresponding safety preservation property (from the left). For example, in the case of sequential composition, Fig. 8(left) indicates that the safety preservation for seq, rephrased as $\mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}\mathsf{safeC}_{(R,G,Q)} \leq \mathsf{safeC}_{(R,G,Q)}$, is proved by strong coinduction, thus the goal takes the form

$$(****)\quad \mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}\mathsf{safeC}_{(R,G,Q)} \leq \mathsf{F}_{(R,G,Q)}(\mathsf{safe}_{(R,G,Q)} \sqcup \mathsf{U}^{\mathsf{seq}}_{(R,G,P,c_2)}\mathsf{safeC}_{(R,G,Q)}).$$

And Fig. 8(right) generalizes this to weak respectfulness, by replacing in (****) the left occurrence of $\mathsf{safe}_{(R,G,Q)}$ with K and the right one with K'. Moreover, here as well as for all the other constructs, the proof of weak respectfulness can be generalized in a completely systematic way from that of the original preservation property (****): In Fig. 5(b)'s coinductive proof of (****), by writing $\mathsf{safe}_{(R,G,Q)}$ and respectively $\mathsf{safe}'_{(R,G,Q)}$ for the occurrences of $\mathsf{safe}_{(R,G,Q)}$ stemming from the left and respectively right of (****), we see that the only properties of this predicate needed in the proof are $\mathsf{safe}_{(R,G,Q)} \leq \mathsf{safe}'_{(R,G,Q)}$ and $\mathsf{safe}_{(R,G,Q)} \leq \mathsf{F}_{(R,G,Q)}\,\mathsf{safe}'_{(R,G,Q)}$—meaning we can replace $\mathsf{safe}_{(R,G,Q)}$ and $\mathsf{safe}'_{(R,G,Q)}$ with arbitrary K and K' such that $K \leq K'$ and $K \leq \mathsf{F}_{(R,G,Q)}\,K'$, yielding a proof of weak respectfulness.

We now come to the coinductive proof of the soundness preservation for while. Fixing R, G, Q, t, c, we take advantage of the coinductive enhancements

for seq and if, and combine them in an operator W that follows the operational semantics of while, namely

$$W = 1_{\mathsf{L}_{(R,G,Q)}} \sqcup (\mathsf{U}^{\mathsf{if}}_{(R,t,\mathsf{seq}\ c\ (\mathsf{while}\ t\ c),\mathsf{done})} \circ (\mathsf{U}^{\mathsf{seq}}_{(R,G,P,\mathsf{while}\ t\ c)}, \lambda K.\ \mathsf{U}^{\mathsf{done}}_{Q})) \,.$$

Combining the weak $\mathsf{F}_{(R,G,Q)}$-respectfulness properties of $U^{\mathsf{done}}_{\_}$, $U^{\mathsf{if}}_{\_}$ and $U^{\mathsf{seq}}_{\_}$ from Fig. 8(right), we obtain that W is weakly respectful whenever $\mathsf{refl}\ G$, $\mathsf{stable}\ Q\ R$ and $\mathsf{stable}\ P\ R$.

The proof by up-to-W coinduction is shown in Fig. 9(b). Comparing it to the inductive proof from Fig. 9(a), similar comments to those in §5.1 apply about the coinductive proof being more direct. As for the enhancement dimension, the benefit of using the inductive hypothesis on the larger term $\mathsf{if}\ t\ (\mathsf{seq}\ c\ (\mathsf{while}\ t\ c))\ \mathsf{done}$ is now achieved by using the coinductive invariant up-to W, i.e., applying W to it in the coinductive goals (4.i) and (4.iii). Admittedly, the more elementary nature of the inductive enhancements can be an advantage. Moreover, the coinductive proof requires anticipation on how the semantics will unfold, to decide which (weakly) respectful operator to use, whereas the inductive proof proceeds analytically, decomposing the goal and using the properties on a need basis—but this can be achieved with the coinductive enhancements too, via the companion (which is also the largest respectful operator) [27,55].

To summarize the enhancement situation abstractly: For operators $F : L \to L$ and $U : L^m \to L$, assume that we were able to prove that U preserves F's greatest fixpoint, i.e., $U\ (\mathsf{gfp}_F, \ldots, \mathsf{gfp}_F) \leq \mathsf{gfp}_F$,

- either inductively, by showing $\forall n.\, U\ (\bigsqcap_m \mathsf{lfp}_{F\sharp} m) \leq \mathsf{lfp}_{F\sharp} n$,
- or coinductively, by showing $U\ (\mathsf{gfp}_F, \ldots, \mathsf{gfp}_F) \leq F\ (U\ (\mathsf{gfp}_F, \ldots, \mathsf{gfp}_F))$.

Then the enhancements proceed as follows:

- for induction, by showing the finer property $\forall n.\, U\ (\mathsf{lfp}_{F\sharp} n, \ldots, \mathsf{lfp}_{F\sharp} n) \leq \mathsf{lfp}_{F\sharp} n$, which we will refer to as *F-approximation-preservation*;
- for coinduction, by showing the finer property that U is weakly F-respectful.

In concrete cases such as our safety preservation properties, for both induction and coinduction the proof of the finer property can be obtained by systematically generalizing the proof of the original one. The two enhancement types are connected as follows.

Thm 10. Assume L is a complete lattice and $F : L \to L$ and $U : L^m \to L$ are monotonic. Then the following hold:
(1) If U is weakly F-respectful, then it is also F-approximation-preserving.
(2) If F is ω-cocontinuous and U is F-approximation-preserving, then up-to-U coinduction is a sound proof method (in the sense of Thm. 9).

Point (1) of the above theorem says that what traditionally makes coinductive enhancements possible, namely (weak) respectfulness, is stronger than what we have identified as making inductive enhancements possible, namely the approximation-preserving property. And point (2) says that what makes the inductive enhancements possible, although a weaker property, still makes the coinductive ones possible.

Preliminaries

We assume **(1)** $P \sqcap (\neg\ (\mathsf{evalT}\ t)) \leq Q\ \wedge\ \mathsf{stable}\ P\ R\ \wedge\ \mathsf{stable}\ Q\ R\ \wedge\ \mathsf{refl}\ G$. We must prove that $c \vDash_{\mathsf{VP}} (P \sqcap (\mathsf{evalT}\ t), R, G, P)$ implies $\mathsf{while}\ t\ c \vDash_{\mathsf{VP}} (P, R, G, Q)$. Expanding the definition of $\vDash_{\mathsf{VP}}$, this reduces to:

Main part

Assuming **(2)** $\forall m, s'.\ P\ s' \wedge \mathsf{evalT}\ t\ s' \longrightarrow \mathsf{safe}_{(R,G,P)}\ m\ (c, s')$, we must prove $\forall n, s.\ P\ s \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{while}\ t\ c, s)$, which we do by induction on n. The base case is again trivial.

For the inductive step, we assume **(IH)** $\forall s.\ P\ s \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{while}\ t\ c, s)$ and must prove $\forall s.\ P\ s \longrightarrow \mathsf{safe}_{(R,G,Q)}\ (n+1)\ (\mathsf{while}\ t\ c, s)$. To this end, we fix s, assume **(3)** $P\ s$, and must prove **(4)** $\mathsf{safe}_{(R,G,Q)}\ (n+1)\ (\mathsf{while}\ t\ c, s)$, which by the definition of $\mathsf{safe}_{(R,G,Q)}$ amounts to:

- **(4.i)** $\forall s'.\ R\ s\ s' \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{while}\ t\ c, s')$
- **(4.ii)** $\mathsf{final}\ (\mathsf{while}\ t\ c, s) \longrightarrow Q\ s$
- **(4.iii)** $\forall d', s'.\ ((\mathsf{while}\ t\ c, s) \Rightarrow (d', s')) \longrightarrow G\ s\ s' \wedge \mathsf{safe}_{(R,G,Q)}\ n\ (d', s')$,

of which the last, thanks to the fact that by the semantics of while its hypothesis means $s = s' \wedge d' = \mathsf{if}\ t\ (\mathsf{seq}\ c\ (\mathsf{while}\ t\ c))\ \mathsf{done}$, reduces to $G\ s\ s \wedge \mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{if}\ t\ (\mathsf{seq}\ c\ (\mathsf{while}\ t\ c))\ \mathsf{done}, s)$, which, since $G\ s\ s$ follows from (1), further reduces to

- **(4.iii')** $\mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{if}\ t\ (\mathsf{seq}\ c\ (\mathsf{while}\ t\ c))\ \mathsf{done}, s)$

Now, from (1), (3) and (IH), we obtain (4.i); and (4.ii) holds because its hypothesis is vacuously false.

Remains to prove (4.iii'), using (IH) and (3)...

Thanks to enhanced preservation for if, (4.iii') reduces to:

- $\mathsf{refl}\ G$, which holds by (1);
- Fixing s', assuming **(5)** $R^*\ s\ s'$, and:
  - Assuming $\neg\ \mathsf{evalT}\ t\ s'$ and showing $\mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{done}, s')$, which follows from (1), (3), (5) and the preservation for done.
  - Assuming **(6)** $\mathsf{evalT}\ t\ s'$ and showing $\mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{seq}\ c\ (\mathsf{while}\ t\ c), s')$, which, thanks to the enhanced preservation for seq, reduces to:
    - (again) $\mathsf{refl}\ G$, holding by (1);
    - $\forall m.\ \mathsf{safe}_{(R,G,P)}\ m\ (c, s')$, following from (2), (3), (6);
    - $\forall s.\ P\ s \longrightarrow \mathsf{safe}_{(R,G,Q)}\ n\ (\mathsf{while}\ t\ c, s)$, i.e., (IH). □

(a) For the counting-based semantics

Preliminaries

We assume **(1)** $P \sqcap (\neg\ (\mathsf{evalT}\ t)) \leq Q\ \wedge\ \mathsf{stable}\ P\ R\ \wedge\ \mathsf{stable}\ Q\ R\ \wedge\ \mathsf{refl}\ G$. We must prove that $c \vDash_{\mathsf{C}} (P \sqcap (\mathsf{evalT}\ t), R, G, P)$ implies $\mathsf{while}\ t\ c \vDash_{\mathsf{C}} (P, R, G, Q)$. Expanding the definition of $\vDash_{\mathsf{C}}$, this reduces to:

Main part

Assuming **(2)** $\forall s'.\ P\ s' \wedge \mathsf{evalT}\ t\ s' \longrightarrow \mathsf{safeC}_{(R,G,P)}\ (c, s')$, we must prove $\forall s.\ P\ s \longrightarrow \mathsf{safeC}_{(R,G,Q)}\ (\mathsf{while}\ t\ c, s)$. To this end, we fix s, c, t, d, assume $P\ s$ and $d = \mathsf{while}\ t\ c$, and must prove $\mathsf{safe}_{(R,G,Q)}\ (d, s)$, which we do by up-to-$W$ coinduction, since W is indeed $F_{(R,G,Q)}$-respectful thanks to (1).

Thus, we assume **(3)** $P\ s$ and $d = \mathsf{while}\ t\ c$, and must show
(4) $\mathsf{F}_{(R,G,Q)}(W\ (\lambda(d, s).\ P\ s \wedge d = \mathsf{while}\ t\ c))\ (d, s)$, which amounts to:

- **(4.i)** $\forall s'.\ R\ s\ s' \longrightarrow W\ (\lambda(d, s).\ P\ s \wedge d = \mathsf{while}\ t\ c))\ (d, s)$,
  which, taking the left disjoint in W, reduces to
- **(4.i')** $\forall s'.\ R\ s\ s' \longrightarrow P\ s' \wedge d = \mathsf{while}\ t\ c$,
  i.e., $\forall s'.\ R\ s\ s' \longrightarrow P\ s'$
- **(4.ii)** $\mathsf{final}\ (d, s) \longrightarrow Q\ s$, i.e., $\mathsf{final}\ (\mathsf{while}\ t\ c, s) \longrightarrow Q\ s$
- **(4.iii)** $\forall d', s'.\ ((d, s) \Rightarrow (d', s')) \longrightarrow G\ s\ s' \wedge W\ (\lambda(d, s).\ P\ s \wedge d = \mathsf{while}\ t\ c))\ (d', s')$,
  which, taking the right disjoint in W, reduces to
- **(4.iii')** $\forall d', s'.\ ((d, s) \Rightarrow (d', s')) \longrightarrow$
  - **(a)** $G\ s\ s' \wedge d' = \mathsf{if}\ t\ (\mathsf{seq}\ c\ (\mathsf{while}\ t\ c))\ \mathsf{done} \wedge$
  - **(b)** $(\forall s'.\ R^*\ s\ s' \wedge \mathsf{evalT}\ t\ s' \longrightarrow \mathsf{safe}_{(R,G,P)}(c, s')) \wedge$
  - **(c)** $(\forall s'.\ R^*\ s\ s' \wedge \neg\,\mathsf{evalT}\ t\ s' \longrightarrow Q\ s') \wedge$
  - **(d)** $(\forall s'.\ P\ s' \longrightarrow P\ s' \wedge d = \mathsf{while}\ t\ c)$

Now, from (1) and (3), we obtain (4.i'); and (4.ii) holds because its hypothesis is vacuously false.
Finally, using $d = \mathsf{while}\ t\ c$ and the semantics of while, the hypothesis of (4.iii') means $s = s' \wedge d' = \mathsf{if}\ t\ (\mathsf{seq}\ c\ (\mathsf{while}\ t\ c))\ \mathsf{done}$, which makes (a) true thanks to (1). Moreover:

- (b) is true thanks to (1), (2) and (3),
- (c) is true thanks to (1) and (3),
- (d) is trivially true (since $d = \mathsf{while}\ t\ c$). □

(b) For the coinductive semantics

Fig. 9: Proofs of soundness of the RG rule for while: for the (inductive) counting-based and coinductive semantics.

6 The Coinduction Approximated Inductively Landscape

Next we look at the scope of coinduction approximated inductively in finer granularity. While so far we only looked at the concrete instances of RG safety operators and a very abstract generalization to operators on lattices, here we will explore a spectrum between these two extremes.

6.1 The rule system format

Thm. 8 captures the general phenomenon behind coinduction approximated inductively, but does not give us a good intuition on its more concrete scope, notably on how it applies to (co)inductive definitions described by sets of rules [2]. To address this, we slightly lower the generality, by focusing on lattices of predicates on a set A, namely $L = (A \to \mathsf{Bool})$ with order defined again by component-wise implication ($K \leq K'$ iff $\forall a \in A.\ K\ a \longrightarrow K'\ a$), and operators given by sets of rules on A, i.e., subsets $Rl \subseteq \mathcal{P}(A) \times A$. So a *rule* on A will be a pair (B, a) where $B \subseteq A$ and $a \in A$; we think of B as the set of the rule's *hypotheses* and of a as the rule's *conclusion*. The associated operator $\mathsf{F}_{Rl} : (A \to \mathsf{Bool}) \to (A \to \mathsf{Bool})$ is defined by applying the rules, namely $\mathsf{F}_{Rl}\ K\ a = (\exists B.\ (B, a) \in Rl \wedge (\forall b \in B.\ K\ b))$. Thus, the notion of a predicate $K : A \to \mathsf{Bool}$ being closed under the rules, namely $\forall (B, a) \in Rl.\ (\forall b \in B.\ K\ b) \longrightarrow K\ a$, is the same as K being a pre-fixpoint of F_{Rl}, i.e., $\mathsf{F}_{Rl}\, K \leq K$. And the notion of K being consistent with (i.e., backwards closed under) the rules, namely $\forall (B, a) \in Rl.\ K\ a \longrightarrow (\forall b \in B.\ K\ b)$, is the same as K being a post-fixpoint of F_{Rl}, i.e. $K \leq \mathsf{F}_{Rl}\, K$. For the least and greatest fixpoints of rule-system operator F_{Rl}, we will write lfp_{Rl} for $\mathsf{lfp}_{\mathsf{F}_{Rl}}$ and gfp_{Rl} for $\mathsf{gfp}_{\mathsf{F}_{Rl}}$.

A first practical observation is that the inductive approximation can be performed intensionally, at the level of rules. Given a set of rules Rl on A, we define $Rl^\sharp$, called the *approximation rules of* Rl, as a set of rules on $\mathbb{N} \times A$, namely $\{(\varnothing, (0, a)) \mid a \in A\} \cup \bigcup_{n \in \mathbb{N}} \{(\{n\} \times B,\ (n+1, a)) \mid (B, a) \in Rl\}$. Thus, $Rl^\sharp$ consists of rules over the same set A but enriched with numeric levels (indexes), i.e., over $\mathbb{N} \times A$. Specifically, it consists of two kinds of rules: (1) any element a of A at level 0 (i.e., paired with 0) becomes an axiom (rule without premises) of the form $(\varnothing, (0, a))$; (2) any rule (B, a) from Rl becomes a rule $(\{n\} \times B, (n+1, a)))$ in $Rl^\sharp$, i.e., adding levels and making sure that the level decreases from $n+1$ to n when applying the rule backwards. Then $\mathsf{F}_{Rl^\sharp}$ and $(\mathsf{F}_{Rl})^\sharp$ are the same (modulo currying), which shifts the discussion from the semantic realm of operators to the (usually) syntactic realm of rules:

Lemma 11. Given $Rl \subseteq \mathcal{P}(A) \times A$ and $K : \mathbb{N} \to A \to \mathsf{Bool}$, we have that $\mathsf{F}_{Rl^\sharp}\ K = (\mathsf{F}_{Rl})^\sharp\ (\lambda n, a.\ K(n, a))$.

And our motivating example predicates are indeed given by rule systems:

Example 12. Both $\mathsf{F}_{(R,G,Q)}$ and $\mathsf{F}^\sharp_{(R,G,Q)}$ are rule-system based. Namely, we have that $\mathsf{F}_{(R,G,Q)} = \mathsf{F}_{Rl_{(R,G,Q)}}$ and $\mathsf{F}^\sharp_{(R,G,Q)} = (\mathsf{F}_{Rl_{(R,G,Q)}})^\sharp = \mathsf{F}_{(Rl_{(R,G,Q)})^\sharp}$,

and therefore $\mathsf{safeC}_{(R,G,Q)} = \mathsf{gfp}_{Rl_{(R,G,Q)}}$ and $\mathsf{safe}_{(R,G,Q)} = \mathsf{lfp}_{(Rl_{(R,G,Q)})^\sharp}$, where $Rl_{(R,G,Q)}$ is the set of pairs $(B,\ (c,s))$ such that $B = \{(c',s') \mid ((c,s) \Rightarrow (c',s'))\} \cup \{(c,s') \mid R\ s\ s'\}$ and $(\mathsf{final}\,(c,s) \longrightarrow Q\ s) \wedge (\forall c', s'.\ ((c,s) \Rightarrow (c',s')) \longrightarrow G\ s\ s')$ holds.

We are ready to describe a more effective criterion for sound approximation. We call a set of rules $Rl \subseteq \mathcal{P}(A) \times A$ *backwards-finite* provided that each $a \in A$ has only a finite number of rules that backwards-apply to it, namely $\forall a \in A$, $\mathsf{finite}\,\{B \subseteq A \mid (B,a) \in Rl\}$. This is a sufficient condition for ω-cocontinuity.

Prop 13. If $Rl \subseteq \mathcal{P}(A) \times A$ is backwards-finite, then F_{Rl} is ω-cocontinuous.

This, together with Lemma 11 and Thm. 8, give us:

Thm 14. Assume that $Rl \subseteq \mathcal{P}(A) \times A$ is backwards-finite. Then, for any $a \in A$, we have that $\mathsf{gfp}_{Rl}\ a$ if and only if $\forall n \in \mathbb{N}.\, \mathsf{lfp}_{Rl^\sharp}\ (n,a)$.

Thus, Thm. 14 is a more concrete depiction of the root cause for the inductive approximation soundness: the backwards finiteness of the defining rules.

6.2 Approximations of bisimilarity

The rule system format has limitations. Notably, it fails to cover the prominent instances of coinduction approximated inductively that occur in the theory of (bi)simulations and (bi)similarity going back to Park and Milner [51,42]—where bisimilarity is first defined coinductively, after which an inductive counterpart is introduced as a technical tool (e.g., for proving logical completeness [26] or domain-theoretic properties [1]). The problem is that the many notions of bisimilarity [42,21,20] cannot be defined via rule systems due to their quantification format. Indeed, the operators underlying rule systems are essentially defined using $\exists\forall$ quantification ("there exists a matching rule such that for all its hypotheses ..."), whereas bisimilarities typically have a $\forall\exists$ format ("for all transitions there exists a matching transition ...").

Furthermore, bisimilarities for systems with names and substitutions, such as late bisimilarity for the π-calculus [43,61], even have a $\forall\exists\forall$ format ("for all input transitions there exists a matching transition such that, for all instantiations of the generic name from the transition..."), as shown in the next example.

Example 15. We assume an Act-labeled transition system $\mathcal{L} = (\mathsf{State}, \mathsf{Act}, \Rightarrow)$ with $\Rightarrow \subseteq \mathsf{State} \times \mathsf{Act} \times \mathsf{State}$ such that there is a set Nm of (channel) names and a renaming operation $\_[\_/\_] : \mathsf{State} \times \mathsf{Nm} \times \mathsf{Nm} \to \mathsf{State}$. The elements $s \in \mathsf{State}$ are called processes, and we think of $s[u/v]$ as the process obtained by the capture-free renaming of u to v. Moreover, we assume a subset $\mathsf{IAct} \subseteq \mathsf{Act}$ of *input actions*, where each input actions has the form $u(v)$ for $u, v \in \mathsf{Nm}$. Now, a *(strong) late bisimulation* is a symmetric relation K on State such that, for all

s_1, s_2, if $K\ s_1\ s_2$ then the following holds:

$$
\begin{array}{l}
(\ \forall\ a, s_1'.\ a \in \mathsf{Act} \setminus \mathsf{IAct} \wedge s_1 \overset{a}{\Rightarrow} s_1' \longrightarrow\ \exists\ s_2'.\ s_2 \overset{a}{\Rightarrow} s_2' \wedge K\ s_1'\ s_2') \wedge \\
(\ \forall\ u, v, s_1'.\ u, v \in \mathsf{Nm} \wedge s_1 \overset{u(v)}{\Rightarrow} s_1' \longrightarrow \qquad\qquad (*) \\
\qquad \exists\ s_2'.\ s_2 \overset{u(v)}{\Rightarrow} s_2' \wedge (\ \forall\ w \in \mathsf{Nm}. K\ (s_1'[w/v])\ (s_2'[w/v]))
\end{array}
$$

The *late bisimilarity* is the largest late bisimulation, i.e., greatest fixpoint of the operator F on relations defined as $F\ K\ (s_1, s_2) \longleftrightarrow K\ (s_2, s_1)\ \wedge\ (*)$.

6.3 An alternating quantifier format

As we discuss next, what makes bisimilarity-defining operators ω-cocontinuous, hence bisimilarities inductively approximable, is the finiteness of the space for existential quantification. Obeying this, any quantifier alternation can be allowed.

We model this by the following predicate. Given a set A and a set of sets Π, of whose elements $P \in \Pi$ we think of as sets of parameters, we define the *existentially-finite alternating quantifier format* predicate $\mathsf{alter} : ((A \to \mathsf{Bool}) \to (A \to \mathsf{Bool})) \to \mathsf{Bool}$, (thus, a predicate on operators in $A \to \mathsf{Bool}$) inductively:

$$\frac{K' : A \to \mathsf{Bool}}{\mathsf{alter}\ (\lambda K.\ K')}(\mathsf{Const}) \qquad \frac{f : A \to A}{\mathsf{alter}\ (\lambda K.\ K \circ f)}(\mathsf{Comp})$$

$$\frac{\begin{array}{c} F : P \to (A \to \mathsf{Bool}) \to (A \to \mathsf{Bool}) \quad T : P \to (A \to \mathsf{Bool}) \\ P \in \Pi \qquad \forall p \in P.\ \mathsf{alter}\ (F\ p) \end{array}}{\mathsf{alter}\ (\lambda\ K, a.\ \forall p \in P.\ T\ p\ a \to F\ p\ K\ a)}(\mathsf{Forall})$$

$$\frac{\begin{array}{c} F : P \to (A \to \mathsf{Bool}) \to (A \to \mathsf{Bool}) \quad T : P \to (A \to \mathsf{Bool}) \\ \forall a.\ \mathsf{finite}\ \{p \mid T\ p\ a\} \qquad P \in \Pi \qquad \forall p \in P.\ \mathsf{alter}\ (F\ p) \end{array}}{\mathsf{alter}\ (\lambda\ K, a.\ \exists p \in P.\ T\ p\ a \wedge F\ p\ K\ a)}(\mathsf{Exists})$$

Thus, $\mathsf{alter}\ F$ says that the operator F (on predicates on A) is obtained by starting with operators that either are constant or just compose their argument predicates with a function on A, and applying (in any order) a sequence of universal and existential quantifiers over parameters, relative to predicates T connecting parameters with the elements of A. Highlighted in the case of existential quantification is that this takes place within a finite space, i.e., the relativization predicate T has finite extension.

The inductive rules defining the alter predicate validate (as being in the "alternating quantifier" format) operators on predicates on a given set A; so these operators map predicates in $A \to \mathsf{Bool}$ to predicates in $A \to \mathsf{Bool}$. Namely, the rules first validate operators that are constant (the (Const) rule) or just compose their input predicate with a function (the (Comp) rule). Moreover, the rules (Forall) and (Exists) validate operators that are obtained from previously validated ones by means of universal or existential quantification over the parameters in some set P, with the additional requirement that existential quantification acts in a finite space. For example, the rule (Forall) says: If, for a given set P

of parameters (belonging to the fixed set of sets Π), we have a P-parameterized operator F (i.e., a function F from P to operators on predicates) and a P-parameterized predicate T (i.e., a function T from P to predicates) such that $F\ p$ has already been validated (i.e., $\mathsf{alter}\ (F\ p)$ holds) for all parameters $p \in P$, then we can validate the operator on predicates that takes any predicate K to the universal quantification of $F\ \_\ K$ over P relativized according to T; namely, when applied to any $a \in A$, this returned predicate says that, for all $p \in P$, $T\ p\ a$ implies $F\ p\ K\ a$.

In the inductive definition of alter, we work with a fixed set of sets Π from which we choose our sets of parameters used for quantification (as opposed to considering arbitrary sets of parameters) in order to make sure that alter is well-defined, i.e., does not rely on large collections. In fact, alter depends on A and Π, and we write $\mathsf{alter}_{A,\Pi}$ when we want to emphasize this dependency.

Note that alter acts on operators on predicates (and not just on predicates) because this is what we are after: identifying operators on predicates that are, by their quantifier format, guaranteed to be ω-continuous (and thus enable the converging inductive approximation of their greatest fixpoints). And indeed, all the operators validated by alter are guaranteed to be ω-cocontinuous:

Thm 16. If $\mathsf{alter}_{A,\Pi}\ F$ holds, then F is ω-cocontinuous; in particular, for any $a \in A$, we have that $\mathsf{gfp}_F\ a$ if and only if $\forall n \in \mathbb{N}.\ \mathsf{lfp}_{F^\sharp}\ n\ a$.

This criterion, which generalizes the rule system format, captures the image-finiteness condition for bisimilarity approximations. Moreover, since alter is also closed under conjunction and disjunction (as a particular case of closure under quantification), we can obtain operators satisfying alter by freely combining positive logical operators, as long as the existentials are finite. In particular, the criterion applies to Example 15, since the π-calculus yields an image-finite labeled transition system. (Note that the additional nested universal quantification in Example 15 takes place in an infinite space (since Nm is usually infinite), but universal quantification is not constrained by the format.) It also generalizes a syntactic criterion implemented in the Dafny program verifier [34] by Leino and Moskal [35]. Their formulation refers to the syntax of predicates definable in Dafny, and involves a syntactic check on a predicate's negation normal form, namely that existential quantification happens over finite types. But their proof (in a technical report cited from [35]) already contains the general apparatus in terms of ω-continuity that we use here.

6.4 Generic RG Separation Logic

We now go back to our starting topic, RG reasoning. The counting-based semantics for RG, which served as our motivating example, is actually a trimmed down version of a semantics for a combination of RG with Separation Logic [65,67,66].

GenRGSep has a process algebra syntax. What is relevant here is that there is a set of actions Act (ranged over by α) which contains a special silent action τ. Moreover, the states s are pairs (s_1, s_2) where s_1 is the local part of the state s_2 is the part shared with the environment (hence subject to change according

$$\mathsf{safeSep}_{(R,G,Q)}\ 0\ (c, s_1, s_2)\ \ \text{(Base)}$$

$$\frac{\begin{array}{l}1.\ \forall s_2', \alpha.\ R\ s_2\ s_2' \longrightarrow \mathsf{safeSep}_{(R,G,Q)}\ n\ (c, s_1, s_2')\\ 2.\ \mathsf{final}\ (c, s_1, s_2) \longrightarrow Q\ (s_1, s_2)\\ 3.\ \forall c', t_1, s_1', s_2', \alpha.\ \alpha \neq \tau \wedge s_1 \preceq t_1 \wedge\\ \qquad\qquad (c, t_1, s_2) \overset{\alpha}{\Rightarrow} (c', s_1', s_2') \longrightarrow G\ s\ s'\\ 4.\ (c, s_1, s_2) \overset{\alpha}{\Rightarrow} (c', s_1', s_2') \longrightarrow \mathsf{safeSep}_{(R,G,Q)}\ n\ (c, s_1', s_2')\\ 5.\ ((c, t_1 + u_1, s_2) \overset{\alpha}{\Rightarrow} (c', s_1', s_2')) \longrightarrow\\ \qquad (\exists t_1'.\ t_1' \# u_1 \wedge s_1' = t_1' + u_1\ \wedge (\alpha = \tau \longrightarrow t_1' = t_1) \wedge\\ \qquad\quad \mathsf{safeSep}_{(R,G,Q)}\ n\ (c, t_1', s_2'))\end{array}}{\mathsf{safeSep}_{(R,G,Q)}\ (n+1)\ (c, s_1, s_2)}\ \text{(Step)}$$

Fig. 10: The inductive counting-based predicate safeSep. Its coinductive counterpart safeSepC is obtained (like before) by removing the items highlighted and interpreting the (Step) rule coinductively.

to the rely relation R). Differently from RG, the execution steps in GenRGSep are labelled by actions, so they have the form $(c, s_1, s_2) \overset{\alpha}{\Rightarrow} (c', s_1', s_2')$. Moreover, local states form *permission algebras* [12], where $+$ denotes the partial semi-group operator and $\#$ denotes the definedness predicate for this operator. We think of $s \# t$ as saying that s and t are non-overlapping (or consistently overlapping) and of $s + t$ as putting together two such non-overlapping components.

The formulation of safety for GenRGSep, predicate safeSep, is shown in Fig. 10. The base case and the inductive hypotheses for the post- and rely-conditions (hypotheses 1 and 2) are essentially the same as those for the RG predicate safe (Fig. 3), but factoring in the distinction between the local and environment-exposed parts of the state. On the other hand, the hypotheses involving steps taken by the command (hypotheses 3–5) are more complex, quantifying universally over extensions of the local state ($s_1 \preceq t_2$ in hypothesis 4) and existentially over non-overlapping partitions of the local state ($t_1' \# u_1$ in the frame-safety hypothesis 5). Details on the rationale for these are provided in [29].

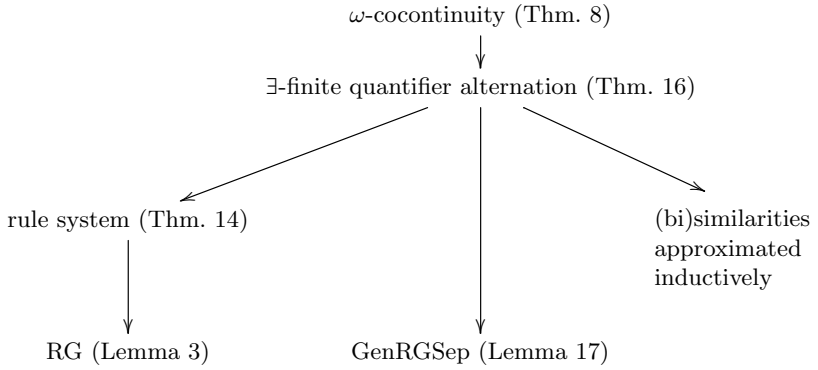
Because hypothesis 5 introduces nested existential quantification (over the component t_1'), this case does not fall under the rule-system format (Thm. 14). However, since all quantification (including the nested one) takes place in a finite space, Thm. 16 applies and produces a coinductive semantics based on a coinductive counterpart safeSepC of safeSep, which is provably equivalent to the inductive semantics—via corresponding versions of Lemma 3 and Thm. 2:

Lemma 17. We have that $\forall n \in \mathbb{N}.\ \mathsf{safeSep}_{(R,G,Q)} n\ (c, s_1, s_2)$ holds if and only if $\mathsf{safeSepC}_{(R,G,Q)}(c, s_1, s_2)$ holds.

Thus, considering the inductive and coinductive satisfaction relations for GenRGSep, $\vDash_{\mathsf{Sep,VP}}\ (P, R, G, Q)$ and $c \vDash_{\mathsf{Sep,C}}$, defined from safeSep and safeSepC just like for RG, we obtain:

Thm 18. We have that $c \vDash_{\mathsf{Sep,VP}}\ (P, R, G, Q)$ if and only if $c \vDash_{\mathsf{Sep,C}}\ (P, R, G, Q)$.

Summary. Below is the hierarchy of the results we discussed pertaining to coinduction approximated inductively, with arrows representing instantiation. ω-cocontinuity is at the top, followed by the existentially-finite alternating quantifier format. The latter is an essential bridge in this hierarchy, because (1) it is an effective criterion, (2) it unifies rule systems and (bi)similarities, and (3) accommodates systems with nested quantification such as GenRGSep.



7 Conclusions and Related Work

Isabelle-supported explorations. The starting point of this work was the goal of formalizing in Isabelle different semantics for Rely-Guarantee and proving them equivalent. The coinductive semantics came from the realization that safety w.r.t. any number of steps in the counting-based semantics is an instance of Kleene-style convergence. Isabelle was instrumental in our proof-mining-like explorations of this realization, which we have described in §5. The Isabelle scripts formalizing this paper's results are provided as supplementary material [15], and details on the formalization are included in our extended technical report [16].

Formal semantics for Rely-Guarantee. Nieto [48] formalized the trace-based semantics in Isabelle; Coleman and Jones [13] developed the reachability based semantics informally, and listed the comparison with Nieto's formal semantics as future work. Vafeiadis and Parkinson [65,67,66] informally, then Jackson et al. formally in Isabelle [29] developed the counting-based semantics for Separation Logic extensions. In particular, Vafeiadis's account from [66], including the more direct soundness proof, was the explicit inspiration for the Jackson et al. paper. We seem to be the first to provide a coinductive semantics of RG, and also to prove the equivalence between previous semantics.

We mostly focused on a simple while language and the basic RG logic, but the coinductive transformation we applied to RG also fits richer languages and logics, as we have illustrated with GenRGSep. Moreover, we focused on Hoare-style RG definitions, rather than algebraic or refinement approaches, which also have a rich literature [6,7,24]. Several RG semantics variations are

reviewed by Van Staden [62] who, taking a trace-based view, classifies them based on whether the guarantee and rely conditions are decoupled.

Counting-free inductive and coinductive accounts of Hoare logic semantics. The idea of defining the partial correctness semantics coinductively has been explored by Wickerson [69,70] in the context of Hoare logic. He proves that a counting-free coinductive semantics for the partial correctness of Hoare triples is equivalent to the standard one. He also proves that the same counting-free definition, but interpreted *inductively* rather than coinductively, yields exactly *total* correctness. His approach to total correctness seems extendable from Hoare logic to Rely-Guarantee. It would be interesting to study how well this compares with the indexed semantics on rule soundness proofs.

Rule systems and formats. Aczel introduced rule systems in his study of inductive definitions [2], considering coinductive definitions only briefly via duality with induction. He singled out deterministic rule systems (where each item can be the conclusion of at most one rule), because these enable defining the least fixpoint by (possibly transfinite) well-founded recursion; deterministic systems are particular cases of backwards-finite rule systems, so they would also ensure ω-cocontinuity. Aczel, and previously Moschovakis [45], were interested in abstract inductive definitions for studying recursively enumerable sets and Gödel encodings. In their theory, inductive predicates specified by positive logic formats play a central role—our alternating quantifier format (generalizing the one implemented in Dafny) is a refinement of the positive logic format. While, in process algebra, formats for operational semantics rules ensuring desirable properties of (bi)similarity have been studied extensively [46], we seem to be the first to look at the formats of the (bi)similarity definitions themselves.

Coinduction approximated inductively. Approximating coinduction by induction is useful for implementing coinduction in theorem provers that initially only support induction. We already mentioned Danfy's coinduction, certified by the Leino-Moskal syntactic criterion. More recently, Mastorou et al. [40] implemented coinduction in Liquid Haskell [68], also relying on inductive approximations. So far their examples do not involve existential quantification, but in general ensuring the relevance of inductive reasoning for the coinductive counterparts will seem to require a criterion similar to Dafny's. Alternatively, if a prover supports transfinite recursion (as is already partially the case with Dafny [36]), then any coinductive predicate can be approximated inductively (without restriction), via a generalization of Kleene's theorem [16, App. C].

While we focused on (co)inductive predicates, similar phenomena occur with datatypes versus codatatypes and recursion versus corecursion. Barr showed that a codatatype can be regarded as a Cauchy completion of a datatype [9,10], and this enables defining functions on codatatypes as convergent approximations of recursive definitions. Matthews [41], and Di Gianantonio and Miculan [19] have built definitional frameworks around these ideas using order-theoretic approximation structures, implemented in Isabelle and Coq respectively.

Usually in the literature (e.g., process algebra bisimilarities, and the aforementioned implementations of coinduction in verifiers) one starts with

the goal of capturing coinductive definitions, and then thinks of how to represent / approximate these inductively. By contrast, here we started with a well-established definition of Rely-Guarantee semantics based on step-indexing, and argued for its "coinductivization". This approach can be applied more generally to step-indexed logical relations [4,3,37], where the numeric index counts the number of steps for which a certain desirable property (such as typing, or in our case RG contract compliance) is guaranteed not to be falsified. Indeed, Nakano's "later" modality [47] used in reasoning with step-based logical relations [17,5] corresponds to our approximating operators $F^\sharp$, and Löb's rule [11] for this modality corresponds to inductive reasoning over approximations; and the approximations are here coinductively sound thanks to the finite branching (and in fact often the determinism) of the transition relation. It would be interesting to explore the connection between Löb's rule and coinduction up-to along the lines of our discussion in §5.2. However, such a coinductive abstraction for step-indexed logical relations would fail as soon as the indexing involves not only numbers (of execution steps) but also other "world" data such as resources *that are mutually dependent* with these numbers—as is the case, for example, when reasoning in higher-order separation logic frameworks such as Iris [32,64].

(Co)inductive enhancements. Notwithstanding the prominently up-to feature of the while construct, the literature on verification based on coinduction up-to did not seem to handle while programs directly like we do, although imperative features were shown to be in the scope of the technique (e.g., [39]).

Our established connection between up-to enhancements of coinduction and corresponding index-wise enhancements of the greatest fixpoint's inductive approximations recommends approximation-preserving operators as a potentially interesting class of operators that are sound for coinduction up-to, which are both compositional and extremely lightweight. In future work, we will study their interaction with parameterized coinduction [27] and second-order reasoning [55] when restricted to ω-cocontinuous operators. Sangiorgi [60] considered a different type of inductive-coinductive connection, adapting up-to techniques to behavioral relations, where coinduction becomes structural induction over the observation contexts. A similar connection has been studied by Goguen and Malcolm in the context of hidden algebra [23].

Acknowledgment. We thank the three anonymous reviewers for their careful reading of the paper and for their insightful comments and suggestions, which helped identify a number of technical typos at key points and improve the overall presentation. Due to space limitations, some suggestions (notably a deeper exploration of the Hoare logic fragment) could not be developed in full; we plan to do so in an extended version to be submitted to a journal. We also thank Rob van Glabbeek for pointing us to his survey papers that systematise the variety of bisimilarity relations used in process algebra. This work was supported by the EPSRC grant EP/X015114/1, "Safe and secure COncurrent programming for adVancEd aRchiTectures (COVERT)". The authors are listed alphabetically, regardless of individual contributions or seniority.

References

1. Abramsky, S.: A domain equation for bisimulation. Inf. Comput. 92(2), 161–218 (1991), https://doi.org/10.1006/inco.1991.9999
2. Aczel, P.: An introduction to inductive definitions. In: Barwise, J. (ed.) Handbook of Mathematical Logic, Studies in Logic and the Foundations of Mathematics, vol. 90, pp. 739–782. Elsevier (1977), https://www.sciencedirect.com/science/article/pii/S0049237X08711200
3. Ahmed, A.: Step-indexed syntactic logical relations for recursive and quantified types. In: European Symposium on Programming (ESOP). Lecture Notes in Computer Science, vol. 3924, pp. 69–83. Springer (2006), https://doi.org/10.1007/11693024_6
4. Appel, A.W., McAllester, D.A.: An indexed model of recursive types for foundational proof-carrying code. ACM Trans. Program. Lang. Syst. 23(5), 657–683 (2001), https://doi.org/10.1145/504709.504712
5. Appel, A.W., Melliès, P., Richards, C.D., Vouillon, J.: A very modal model of a modern, major, general type system. In: Hofmann, M., Felleisen, M. (eds.) Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007. pp. 109–122. ACM (2007), https://doi.org/10.1145/1190216.1190235
6. Armstrong, A., Gomes, V.B.F., Struth, G.: Algebraic principles for rely-guarantee style concurrency verification tools. In: Jones, C., Pihlajasaari, P., Sun, J. (eds.) FM 2014: Formal Methods. pp. 78–93. Springer International Publishing, Cham (2014)
7. Armstrong, A., Gomes, V.B.F., Struth, G.: Algebras for program correctness in isabelle/hol. In: Höfner, P., Jipsen, P., Kahl, W., Müller, M.E. (eds.) Relational and Algebraic Methods in Computer Science - 14th International Conference, RAMiCS 2014, Marienstatt, Germany, April 28-May 1, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8428, pp. 49–64. Springer (2014), https://doi.org/10.1007/978-3-319-06251-8_4
8. Ballarin, C.: Tutorial to Locales and Locale Interpretation. In: Contribuciones Científicas en Honor de Mirian Andrés Gómez, pp. 123–140. University of Rioja, Spain (2010), online at https://dialnet.unirioja.es/servlet/articulo?codigo=3216664
9. Barr, M.: Terminal coalgebras in well-founded set theory. Theor. Comput. Sci. 114(2), 299–315 (1993), https://doi.org/10.1016/0304-3975(93)90076-6
10. Barr, M.: Additions and corrections to "terminal coalgebras in well-founded set theory". Theor. Comput. Sci. 124(1), 189–192 (1994), https://doi.org/10.1016/0304-3975(94)90060-4
11. Boolos, G.S.: The Logic of Provability. Cambridge University Press, Cambridge and New York (1993), https://www.cambridge.org/core/books/logic-of-provability/F1549530F91505462083CE2FEB6444AA
12. Calcagno, C., O'Hearn, P.W., Yang, H.: Local action and abstract separation logic. In: 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings. pp. 366–378. IEEE Computer Society (2007), https://doi.org/10.1109/LICS.2007.30
13. Coleman, J.W., Jones, C.B.: A structural proof of the soundness of rely/guarantee rules. J. Log. Comput. 17(4), 807–841 (2007), https://doi.org/10.1093/logcom/exm030

14. Cousot, P., Cousot, R.: Constructive versions of Tarski's fixed point theorems. Pacific Journal of Mathematics 82(1), 43 – 57 (1979)
15. Derrick, J., Edmonds, C., Popescu, A., Wright, J.: Formal Isabelle development associated with this paper. https://zenodo.org/records/18319366 (2026)
16. Derrick, J., Edmonds, C., Popescu, A., Wright, J.: Rely-Guarantee Is Coinductive: A Proof-Centered Investigation of Inductively Approximated Coinduction. Extended Technical Report. https://www.andreipopescu.uk/TechReports/ESOP2026/TR_RelyGuarantee_is_Coinductive.pdf (2026)
17. Dreyer, D., Ahmed, A., Birkedal, L.: Logical step-indexed logical relations. Log. Methods Comput. Sci. 7(2) (2011), https://doi.org/10.2168/LMCS-7(2:16)2011
18. Gavran, I., Niksic, F., Kanade, A., Majumdar, R., Vafeiadis, V.: Rely/guarantee reasoning for asynchronous programs. In: Aceto, L., de Frutos-Escrig, D. (eds.) 26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015. LIPIcs, vol. 42, pp. 483–496. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015), https://doi.org/10.4230/LIPIcs.CONCUR.2015.483
19. Gianantonio, P.D., Miculan, M.: A unifying approach to recursive and co-recursive definitions. In: Geuvers, H., Wiedijk, F. (eds.) Types for Proofs and Programs, Second International Workshop, TYPES 2002, Berg en Dal, The Netherlands, April 24-28, 2002, Selected Papers. Lecture Notes in Computer Science, vol. 2646, pp. 148–161. Springer (2002), https://doi.org/10.1007/3-540-39185-1_9
20. van Glabbeek, R.J.: The linear time - branching time spectrum II. In: Best, E. (ed.) CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings. Lecture Notes in Computer Science, vol. 715, pp. 66–81. Springer (1993), https://doi.org/10.1007/3-540-57208-2_6
21. van Glabbeek, R.J.: The linear time - branching time spectrum I. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 3–99. North-Holland / Elsevier (2001), https://doi.org/10.1016/b978-044482830-9/50019-9
22. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics (extended abstract). In: Ritter, G.X. (ed.) Information Processing 89, Proceedings of the IFIP 11th World Computer Congress, San Francisco, USA, August 28 - September 1, 1989. pp. 613–618. North-Holland/IFIP (1989)
23. Goguen, J.A., Malcolm, G.: Hidden coinduction: behavioural correctness proofs for objects. Math. Struct. Comput. Sci. 9(3), 287–319 (1999), http://journals.cambridge.org/action/displayAbstract?aid=44821
24. Hayes, I.J.: Generalised rely-guarantee concurrency: an algebraic foundation. Form. Asp. Comput. 28(6), 1057–1078 (Nov 2016), https://doi.org/10.1007/s00165-016-0384-0
25. Hayes, I.J., Jones, C.B.: A guide to rely/guarantee thinking. In: Bowen, J.P., Liu, Z., Zhang, Z. (eds.) Engineering Trustworthy Software Systems - Third International School, SETSS 2017, Chongqing, China, April 17-22, 2017, Tutorial Lectures. Lecture Notes in Computer Science, vol. 11174, pp. 1–38. Springer (2017), https://doi.org/10.1007/978-3-030-02928-9_1
26. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. J. ACM 32(1), 137–161 (1985), https://doi.org/10.1145/2455.2460
27. Hur, C., Neis, G., Dreyer, D., Vafeiadis, V.: The power of parameterization in coinductive proof. In: Giacobazzi, R., Cousot, R. (eds.) The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL

'13, Rome, Italy - January 23 - 25, 2013. pp. 193–206. ACM (2013), https://doi.org/10.1145/2429069.2429093
28. Ishtiaq, S.S., O'Hearn, P.W.: BI as an assertion language for mutable data structures. In: Hankin, C., Schmidt, D. (eds.) Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001. pp. 14–26. ACM (2001), https://doi.org/10.1145/360204.375719
29. Jackson, V., Murray, T., Rizkallah, C.: A generalised union of rely-guarantee and separation logic using permission algebras. In: Bertot, Y., Kutsia, T., Norrish, M. (eds.) 15th International Conference on Interactive Theorem Proving, ITP 2024, September 9-14, 2024, Tbilisi, Georgia. LIPIcs, vol. 309, pp. 23:1–23:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024), https://doi.org/10.4230/LIPIcs.ITP.2024.23
30. Jones, C.B.: Development Methods for Computer Programs including a Notion of Interference. Ph.D. thesis, Oxford University (Jun 1981), http://www.cs.ox.ac.uk/files/9025/PRG-25.pdf, printed as: Programming Research Group, Technical Monograph 25
31. Jones, C.B.: Specification and design of (parallel) programs. In: Mason, R.E.A. (ed.) Information Processing 83, Proceedings of the IFIP 9th World Computer Congress, Paris, France, September 19-23, 1983. pp. 321–332. North-Holland/IFIP (1983)
32. Jung, R., Swasey, D., Sieczkowski, F., Svendsen, K., Turon, A., Birkedal, L., Dreyer, D.: Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In: Rajamani, S.K., Walker, D. (eds.) Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015. pp. 637–650. ACM (2015), https://doi.org/10.1145/2676726.2676980
33. Kozen, D., Silva, A.: Practical coinduction. Math. Struct. Comput. Sci. 27(7), 1132–1152 (2017), https://doi.org/10.1017/S0960129515000493
34. Leino, K.R.M.: Dafny: An automatic program verifier for functional correctness. In: Clarke, E.M., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers. Lecture Notes in Computer Science, vol. 6355, pp. 348–370. Springer (2010), https://doi.org/10.1007/978-3-642-17511-4_20
35. Leino, K.R.M., Moskal, M.: Co-induction simply - automatic co-inductive proofs in a program verifier. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8442, pp. 382–398. Springer (2014), https://doi.org/10.1007/978-3-319-06410-9_27
36. Leino, K.R.M., Tristan, J.B.: Dafny power user: Working with coinduction, extreme predicates, and ordinals (2023), technical report. https://leino.science/papers/krml285.html
37. Leroy, X.: Step carefully: Step-indexing techniques. Lecture Notes, Collège de France (Jan 2019), https://xavierleroy.org/CdF/2018-2019/8.pdf
38. Leroy, X., Grall, H.: Coinductive big-step operational semantics. Inf. Comput. 207(2), 284–304 (2009), https://doi.org/10.1016/j.ic.2007.12.004
39. Madiot, J., Pous, D., Sangiorgi, D.: Modular coinduction up-to for higher-order languages via first-order transition systems. Log. Methods Comput. Sci. 17(3) (2021), https://doi.org/10.46298/lmcs-17(3:25)2021

40. Mastorou, L., Papaspyrou, N., Vazou, N.: Coinduction inductively: mechanizing coinductive proofs in liquid haskell. In: Polikarpova, N. (ed.) Haskell '22: 15th ACM SIGPLAN International Haskell Symposium, Ljubljana, Slovenia, September 15 - 16, 2022. pp. 1–12. ACM (2022), https://doi.org/10.1145/3546189.3549922
41. Matthews, J.: Recursive function definition over coinductive types. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin-Mohring, C., Théry, L. (eds.) Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1690, pp. 73–90. Springer (1999), https://doi.org/10.1007/3-540-48256-3_6
42. Milner, R.: Communication and concurrency. Prentice Hall (1989)
43. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, II. Inf. Comput. 100(1), 41–77 (1992), https://doi.org/10.1016/0890-5401(92)90009-5
44. Momigliano, A., Pientka, B., Thibodeau, D.: A case study in programming coinductive proofs: Howe's method. Math. Struct. Comput. Sci. 29(8), 1309–1343 (2019), https://doi.org/10.1017/S0960129518000415
45. Moschovakis, Y.N.: Elementary induction on abstract structures (Studies in logic and the foundations of mathematics). American Elsevier Pub. Co (1974)
46. Mousavi, M.R., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. Theor. Comput. Sci. 373(3), 238–272 (2007), https://doi.org/10.1016/j.tcs.2006.12.019
47. Nakano, H.: A modality for recursion. In: 15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000. pp. 255–266. IEEE Computer Society (2000), https://doi.org/10.1109/LICS.2000.855774
48. Nieto, L.P.: The Rely-Guarantee method in Isabelle/HOL. In: Degano, P. (ed.) Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2618, pp. 348–362. Springer (2003), https://doi.org/10.1007/3-540-36575-3_24
49. Nipkow, T., Paulson, L., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
50. O'Hearn, P.W., Reynolds, J.C., Yang, H.: Local reasoning about programs that alter data structures. In: Fribourg, L. (ed.) Computer Science Logic, 15th International Workshop, CSL 2001. 10th Annual Conference of the EACSL, Paris, France, September 10-13, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2142, pp. 1–19. Springer (2001), https://doi.org/10.1007/3-540-44802-0_1
51. Park, D.M.R.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.) Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23-25, 1981, Proceedings. Lecture Notes in Computer Science, vol. 104, pp. 167–183. Springer (1981), https://doi.org/10.1007/BFb0017309
52. Paulson, L.C.: A fixedpoint approach to implementing (co)inductive definitions. In: Bundy, A. (ed.) Automated Deduction - CADE-12, 12th International Conference on Automated Deduction, Nancy, France, June 26 - July 1, 1994, Proceedings. Lecture Notes in Computer Science, vol. 814, pp. 148–161. Springer (1994), https://doi.org/10.1007/3-540-58156-1_11
53. Pierce, B.C.: Types and Programming Languages. MIT Press (2002)
54. Pous, D.: Complete lattices and up-to techniques. In: Shao, Z. (ed.) Programming Languages and Systems, 5th Asian Symposium, APLAS 2007, Singapore, November 29-December 1, 2007, Proceedings. Lecture Notes in Computer

Science, vol. 4807, pp. 351–366. Springer (2007), https://doi.org/10.1007/978-3-540-76637-7_24
55. Pous, D.: Coinduction all the way up. In: Grohe, M., Koskinen, E., Shankar, N. (eds.) Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016. pp. 307–316. ACM (2016), https://doi.org/10.1145/2933575.2934564
56. Pous, D.: Coinduction: Automata, formal proof, companions (invited paper). In: Roggenbach, M., Sokolova, A. (eds.) 8th Conference on Algebra and Coalgebra in Computer Science, CALCO 2019, June 3-6, 2019, London, United Kingdom. LIPIcs, vol. 139, pp. 4:1–4:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019), https://doi.org/10.4230/LIPIcs.CALCO.2019.4
57. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: 17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings. pp. 55–74. IEEE Computer Society (2002), https://doi.org/10.1109/LICS.2002.1029817
58. Rutten, J.J.M.M.: Universal coalgebra: a theory of systems. Theor. Comput. Sci. 249(1), 3–80 (2000), https://doi.org/10.1016/S0304-3975(00)00056-6
59. Sangiorgi, D.: On the proof method for bisimulation (extended abstract). In: Wiedermann, J., Hájek, P. (eds.) Mathematical Foundations of Computer Science 1995, 20th International Symposium, MFCS'95, Prague, Czech Republic, August 28 - September 1, 1995, Proceedings. Lecture Notes in Computer Science, vol. 969, pp. 479–488. Springer (1995), https://doi.org/10.1007/3-540-60246-1_153
60. Sangiorgi, D.: An abstract account of up-to techniques for inductive behavioural relations. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. REoCAS Colloquium in Honor of Rocco De Nicola - 12th International Symposium, ISoLA 2024, Crete, Greece, October 27-31, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 15219, pp. 62–74. Springer (2024), https://doi.org/10.1007/978-3-031-73709-1_5
61. Sangiorgi, D., Walker, D.: The Pi-Calculus - a theory of mobile processes. Cambridge University Press (2001)
62. van Staden, S.: On rely-guarantee reasoning. In: Hinze, R., Voigtländer, J. (eds.) Mathematics of Program Construction - 12th International Conference, MPC 2015, Königswinter, Germany, June 29 - July 1, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9129, pp. 30–49. Springer (2015), https://doi.org/10.1007/978-3-319-19797-5_2
63. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5, 285–309 (1955), https://api.semanticscholar.org/CorpusID:13651629
64. Timany, A., Krebbers, R., Dreyer, D., Birkedal, L.: A logical approach to type soundness. J. ACM 71(6), 40:1–40:75 (2024), https://doi.org/10.1145/3676954
65. Vafeiadis, V.: Modular fine-grained concurrency verification. Tech. Rep. UCAM-CL-TR-726, University of Cambridge, Computer Laboratory (Jul 2008), https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-726.pdf
66. Vafeiadis, V.: Concurrent separation logic and operational semantics. In: Mislove, M.W., Ouaknine, J. (eds.) Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics, MFPS 2011, Pittsburgh, PA, USA, May 25-28, 2011. Electronic Notes in Theoretical Computer Science, vol. 276, pp. 335–351. Elsevier (2011), https://doi.org/10.1016/j.entcs.2011.09.029
67. Vafeiadis, V., Parkinson, M.J.: A marriage of rely/guarantee and separation logic. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007 - Concurrency Theory, 18th

International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4703, pp. 256–271. Springer (2007), https://doi.org/10.1007/978-3-540-74407-8_18
68. Vazou, N., Seidel, E., Vytiniotis, D., Peyton Jones, S., Jhala, R.: Liquid types. In: Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 159–170. ACM (2014)
69. Wickerson, J.: Partial and total correctness as greatest and least fixed points. https://johnwickerson.github.io/papers/Partial_Total_Correctness_As_Fixed_Points.pdf (2009), Unpublished technical note.
70. Wickerson, J.: Partial and total correctness as greatest and least fixed points. https://johnwickerson.wordpress.com/2009/11/25/partialtotal/ (2009), Blog post.
71. Xia, L., Zakowski, Y., He, P., Hur, C., Malecha, G., Pierce, B.C., Zdancewic, S.: Interaction trees: representing recursive and impure programs in coq. Proc. ACM Program. Lang. 4(POPL), 51:1–51:32 (2020), https://doi.org/10.1145/3371119
72. Xu, Q., de Roever, W.P., He, J.: The rely-guarantee method for verifying shared variable concurrent programs. Formal Aspects Comput. 9(2), 149–174 (1997), https://doi.org/10.1007/BF01211617