First-Order Guarded Coinduction in Coq

Łukasz Czajka
Faculty of Informatics, TU Dortmund University, Germany
lukaszcz@mimuw.edu.pl

Abstract

We introduce two coinduction principles and two proof translations which, under certain conditions, map coinductive proofs that use our principles to guarded Coq proofs. The first principle provides an "operational" description of a proof by coinduction, which is easy to reason with informally. The second principle extends the first one to allow for direct proofs by coinduction of statements with existential quantifiers and multiple coinductive predicates in the conclusion. The principles automatically enforce the correct use of the coinductive hypothesis. We implemented the principles and the proof translations in a Coq plugin.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Logic and verification

Keywords and phrases coinduction, Coq, guardedness, corecursion

Digital Object Identifier 10.4230/LIPIcs.ITP.2019.14

Related Version A full version of the paper including the appendix is available at `https://www.mimuw.edu.pl/~lukaszcz/focoind.pdf`.

Supplement Material The Coq plugin is available at `https://github.com/lukaszcz/coinduction`.

1 Introduction

Coinduction has been studied for several decades now, and is being used increasingly often in practice. Most formal coinduction principles are based on the lattice-theoretic Knaster-Tarski fixpoint theorem [19, 18], on category theory [13, 16], on a syntactic description of legal proofs [5, 10], or on corecursors [15, 9]. Arguably, these principles are not well-suited for informal reasoning, and complex coinductive arguments are difficult to verify without a formalisation or a tedious reformulation.

Induction is dual to coinduction and it has dual lattice-theoretic and category-theoretic formulations, but informal proofs by induction normally follow an "operational" understanding of how to apply the inductive hypothesis: an argument to the inductive hypothesis must decrease in an appropriate sense. This informal understanding is reflected in Coq's induction principles and associated tactics. We propose a formal coinduction principle based on a dual (in an informal sense) "operational" understanding of how to use the coinductive hypothesis: the result must increase in an appropriate sense. This principle overcomes a weakness of Coq's current setup, where proofs built automatically by run-of-the-mill tactics may later be rejected by the type-checker on the grounds that they are not guarded.

A reader familiar with research in coinduction will probably notice a similarity between our first coinduction principle and some prior work, e.g., the principle from [14, 4.10] or the work on sized types [3, 2, 17, 1, 11] (see Remark 3.1). A contribution of this paper is to show that a principle of this kind may, to some extent, be already implemented in Coq's type theory, with the proofs translated directly to guarded Coq proof terms. From this point of view, Coq's guardedness criterion turns out to essentially be a syntactic description of the shape of normal forms of proofs obtainable using our principle. Gimenez [10, Theorem 8] already showed that his guardedness criterion is equivalent, in terms of definable functions, to corecursors in the style of [15, 9], but these are not convenient to use directly.

We also propose a second coinduction principle which extends the first one to allow for direct coinductive proofs of statements with existential quantifiers and multiple coinductive predicates in the conclusion.

The first coinduction principle may be implemented in Coq relatively seamlessly, with only small restrictions of limited practical significance. The situation is less satisfactory with the second principle. Significantly stronger restrictions are required, and the theoretical guarantees are weak. Nonetheless, the implementation is still useful. It covers a common pattern of proofs of existential statements that occur, e.g., in proofs about infinitary lambda-calculus [6]. Moreover, the difficulties with the implementation of the second principle seem to be caused by the limitations of Coq's type theory rather than by some more fundamental problems (see Remark 4.10).

The Paco library [12] achieves similar practical objectives to the first coinduction principle from our Coq plugin, but its methods are orthogonal to ours. It is based on parameterised coinduction – an extension of the common lattice-theoretic coinduction principle. It replaces Coq's `cofix` and requires the user to reformulate the definitions of their coinductive predicates using constructs from the library. In contrast, our approach is to translate the proofs obtained with our principle directly to guarded Coq proofs, which does not require any reformulation of the coinductive predicates. The translation approach has some practical disadvantages (e.g. Coq still wastes time on doing the guardedness checks), but our contribution is more in proposing a principle which may be considered an approximate semantic counterpart to Coq's syntactic guardedness check, thus opening an interesting line for future work.

Our principles are partly inspired by the explanations in [7] of how to elaborate proofs by coinduction to non-coinductive proofs in set theory.

2 Informal description

In this section, we informally state two coinduction principles and illustrate their use with a few examples of coinductive proofs. In the rest of this paper, we investigate to what extent and under which assumptions the principles may be implemented in Coq. The purpose of this section is to give an informal, illustrative introduction.

A (co)inductive type is given by its constructors, presented as, e.g.,

$$\texttt{Stream}(A : *) : * := \texttt{cons} : A \to \texttt{Stream}\, A \to \texttt{Stream}\, A$$

where $*$ denotes the sort of types. Above A is a *parameter* and $* \to *$ is the *arity* of `Stream`. The types of constructors implicitly quantify over the parameters, i.e., the type of `cons` above is $\forall A : *. A \to \texttt{Stream}\, A \to \texttt{Stream}\, A$. In the presentation we leave the parameter A implicit. Intuitively, a coinductive type consist of all possibly infinite objects built using the constructors, while an inductive type consists only of the finite ones.

Statements (logical formulas) are represented by dependent types. (Co)inductive predicates are represented by dependent (co)inductive types, e.g., the coinductive type

$$\begin{aligned}&\texttt{EqSt}(A : *) : \texttt{Stream}\, A \to \texttt{Stream}\, A \to * :=\\ &\quad \texttt{eqst} : \forall x : A. \forall s_1, s_2 : \texttt{Stream}\, A.\\ &\qquad\qquad \texttt{EqSt}\, A\, s_1\, s_2 \to \texttt{EqSt}\, A\, (\texttt{cons}\, x\, s_1)\, (\texttt{cons}\, x\, s_2)\end{aligned}$$

defines equality (bisimilarity) on streams. We use the words "statement" and "predicate" when we want to emphasise the logical interpretation of dependent types.

To state the coinduction principles, for each coinductive type I we need to define two associated types: the red type I^r and the green type I^g. Here we only informally describe them. The types I^r and I^g have the same parameters and the same arity as I and satisfy the following two properties. Below, we assume $I\, s_1 \ldots s_k : *$.

- The red type I^r is a fresh type symbol such that any value in $Is_1 \dots s_k$ or in $I^g s_1 \dots s_k$ may be (implicitly) converted into a value in $I^r s_1 \dots s_k$.
- The green type I^g is an inductive type such that for every constructor

  $$c : \forall x_1 : \tau_1 \dots \forall x_n : \tau_n . I s_1 \dots s_k$$

  of I there is a corresponding green constructor

  $$c^g : \forall x_1 : \tau_1[I^r/I] \dots \forall x_n : \tau_n[I^r/I] . I^g s_1 \dots s_k .$$

Nothing else is known about I^r and I^g. In particular, case analysis on values in $I^r s_1 \dots s_k$ is not possible. Note that any value v in $Is_1 \dots s_k$ may be converted into a value in $I^g s_1 \dots s_k$, by doing case analysis on v, in each case converting subterms of type $Is'_1 \dots s'_k$ to values in $I^r s'_1 \dots s'_k$, and then applying the corresponding green constructor.

▶ **Example 2.1.** For the type of streams `Stream` the green type $\mathtt{Stream}^g$ is:

$$\mathtt{Stream}^g (A : *) : * := \mathtt{cons}^g : A \to \mathtt{Stream}^r\ A \to \mathtt{Stream}^g\ A$$

For the bisimilarity `EqSt` on streams the green type $\mathtt{EqSt}^g$ is:

$$\begin{aligned} &\mathtt{EqSt}^g (A : *) : \mathtt{Stream}\ A \to \mathtt{Stream}\ A \to * := \\ &\quad \mathtt{eqst}^g : \forall x : A . \forall s_1, s_2 : \mathtt{Stream}\ A . \\ &\qquad\qquad \mathtt{EqSt}^r\ A\ s_1\ s_2 \to \mathtt{EqSt}^g\ A\ (\mathtt{cons}\ x\ s_1)\ (\mathtt{cons}\ x\ s_2) \end{aligned}$$

In a type $\varphi = \forall x_1 : \tau_1 \dots \forall x_n : \tau_n . I s_1 \dots s_k$ the type $Is_1 \dots s_k$ is the *target* and I is the *target (co)inductive predicate*. We write $\varphi(I')$ for φ with the target (co)inductive predicate replaced by I'. So $\varphi(I') = \forall x_1 : \tau_1 \dots \forall x_n : \tau_n . I' s_1 \dots s_k$. Note that the substitution of I' in $\varphi(I')$ leaves the occurrences of I in $\tau_1, \dots, \tau_n$ intact.

We restrict our coinduction principles to first-order statements and first-order (co)inductive types. First-order types will be defined precisely in the next section. Essentially, we need to disallow quantification over types and type constructors, excepting parameters of (co)inductive types. Also, for the actual implementation in Coq some further restrictions are needed, especially for the second principle.

▶ **Principle 1** (First coinduction principle – informal). *Let I be a coinductive type and $\varphi(I)$ a first-order statement. If $\varphi(I^r)$ implies $\varphi(I^g)$ then $\varphi(I)$ holds.*

The statement $\varphi(I^r)$ is the *coinductive hypothesis*, and $\varphi(I^g)$ is the *coinductive claim*. Hence, a proof by coinduction shows the coinductive claim under the assumption of the coinductive hypothesis. Intuitively, the red type I^r in the antecedent of the implication $\varphi(I^r) \to \varphi(I^g)$ ensures that a proof of $I^r s_1 \dots s_k$ obtained from the coinductive hypothesis cannot be analysed in any way, or used with previously proven lemmas about I. The green type I^g in the succedent ensures that a constructor must always be applied to a proof obtained from the coinductive hypothesis, i.e., it ensures productivity and prohibits concluding with the coinductive hypothesis directly. In this way, we ensure semantic guardedness of any proof of $\varphi(I^r) \to \varphi(I^g)$, i.e., the guarded use of the coinductive hypothesis $\varphi(I^r)$. Such a proof may then be translated into a guarded coinductive proof of $\varphi(I)$.

▶ **Example 2.2.** We show that the bisimilarity `EqSt` on streams is an equivalence relation. We write $s_1 \approx s_2$ instead of $\mathtt{EqSt}\ A\ s_1\ s_2$, and analogously with $\approx^r$ and $\approx^g$. We omit the type parameter A when irrelevant.

Using the first coinduction principle, we prove by coinduction that $\approx$ is reflexive. The coinductive hypothesis is: $s \approx^r s$ for all streams s. We need to show $s \approx^g s$ for all streams s. Let s be a stream. We have $s = \mathtt{cons}\, x\, s'$. By the coinductive hypothesis $s' \approx^r s'$. Hence $\mathtt{cons}\, x\, s' \approx^g \mathtt{cons}\, x\, s'$ by the definition of $\approx^g$.

We now prove by coinduction that $\approx$ is symmetric. The coinductive hypothesis is: for all streams s_1, s_2, if $s_1 \approx s_2$ then $s_2 \approx^r s_1$. Let s_1, s_2 be streams such that $s_1 \approx s_2$. Then $s_1 = \mathtt{cons}\, x\, s_1'$ and $s_2 = \mathtt{cons}\, x\, s_2'$ with $s_1' \approx s_2'$, by the definition of $\approx$. By the coinductive hypothesis $s_2' \approx^r s_1'$. Hence $\mathtt{cons}\, x\, s_2' \approx^g \mathtt{cons}\, x\, s_1'$ by the definition of $\approx^g$.

Finally, we prove transitivity of $\approx$ by coinduction. Let s_1, s_2, s_3 be streams such that $s_1 \approx s_2$ and $s_2 \approx s_3$. Then $s_1 = \mathtt{cons}\, x\, s_1'$, $s_2 = \mathtt{cons}\, x\, s_2'$ and $s_3 = \mathtt{cons}\, x\, s_3'$ with $s_1' \approx s_2'$ and $s_2' \approx s_3'$, by the definition of $\approx$. By the coinductive hypothesis $s_1' \approx^r s_3'$. Hence $\mathtt{cons}\, x\, s_1' \approx^g \mathtt{cons}\, x\, s_3'$ by the definition of $\approx^g$.

The first coinduction principle requires the target of the statement being proved to be a single coinductive predicate. This is in line with most previous work on coinduction. We will now informally state the second coinduction principle which enables direct coinductive proofs of statements with more complex targets.

Conjunction (product) $\wedge$, usually written in infix notation, may be defined by:

$$\wedge(A : *)(B : *) : * := \mathtt{conj} : A \to B \to A \wedge B$$

Existential quantification (dependent sum) may also be defined as an inductive type:

$$\mathtt{ex}(A : *)(P : A \to *) : * := \mathtt{ex\_intro} : \forall x : A. \forall p : Px. \mathtt{ex}AP$$

We usually write $\exists x : A.t$ instead of $\mathtt{ex}A(\lambda x : A.t)$.

We consider statements

$$\varphi = \forall x_1 : \tau_1 \ldots \forall x_m : \tau_m. \exists y : I t_1 \ldots t_p. I_1 s_1^1 \ldots s_{k_1}^1 y \wedge \ldots \wedge I_n s_1^n \ldots s_{k_n}^n y$$

where y does not occur in s_i^j. Thus the target is a single existential quantification on a value of a coinductive type followed by a conjunction of n coinductive predicates ($n \geq 1$) which depend on the existentially quantified variable. We write $\varphi(I'; I_1', \ldots, I_n')$ for φ with $I, I_1, \ldots, I_n$ in the target replaced by $I', I_1', \ldots, I_n'$ respectively (other occurrences of $I, I_1', \ldots, I_n'$ in $\tau_1, \ldots, \tau_m$ are not affected). For the sake of simplicity, we require y to always be the last argument of I_i, but the extension to the general case is straightforward.

The problem now is that changing the type of y will result in the whole statement being no longer well-typed. We thus introduce dependent red and green types by modifying the definitions of the red and the green types. We replace the last coinductive type in the arity with the corresponding red or green type, respectively, and for green types we also modify the types of the constructors accordingly. The definition of dependent red and green types works only for certain coinductive types. The precise conditions will be given later.

▶ **Example 2.3.** For the bisimilarity `EqSt` on streams the dependent red type has the type

$$\mathtt{EqSt}^r : \forall A : *. \mathtt{Stream}\, A \to \mathtt{Stream}^r A \to *$$

and the dependent green type $\mathtt{EqSt}^g$ is:

$$\begin{aligned} &\mathtt{EqSt}^g(A : *) : \mathtt{Stream}\, A \to \mathtt{Stream}^g A \to * := \\ &\quad \mathtt{eqst}^g : \forall x : A. \forall s_1 : \mathtt{Stream}\, A. \forall s_2^r : \mathtt{Stream}^r A. \\ &\qquad \mathtt{EqSt}^r\, A\, s_1\, s_2^r \to \mathtt{EqSt}^g\, A\, (\mathtt{cons}\, x\, s_1)\, (\mathtt{cons}^g\, x\, s_2^r) \end{aligned}$$

We can now informally state the second coinduction principle for statements with existential quantification in the target. When we write $\varphi(I^r; I_1^r, \ldots, I_n^r)$ we assume I^r is an (ordinary) red type and $I_1^r, \ldots, I_n^r$ are dependent red types. Analogously with $\varphi(I^g; I_1^g, \ldots, I_n^g)$.

▶ **Principle 2** (Second coinduction principle – informal). *Let $I, I_1, \ldots, I_n$ be coinductive types and $\varphi(I; I_1, \ldots, I_n)$ a first-order statement. If $\varphi(I^r; I_1^r, \ldots, I_n^r)$ implies $\varphi(I^g; I_1^g, \ldots, I_n^g)$ then $\varphi(I; I_1, \ldots, I_n)$ holds.*

▶ **Example 2.4.** Consider the following coinductive type of infinite terms.

$$\texttt{term} : * := C : \texttt{nat} \to \texttt{term} \mid A : \texttt{term} \to \texttt{term} \mid B : \texttt{term} \to \texttt{term} \to \texttt{term}$$

We define a parallel reduction relation $\Rightarrow$ on such terms, written in infix notation.

$$\begin{aligned} \Rightarrow : \texttt{term} \to \texttt{term} \to * \quad := \quad & r_C : \forall i : \texttt{nat}. Ci \Rightarrow Ci \\ & \mid r_A : \forall tt'. t \Rightarrow t' \to At \Rightarrow At' \\ & \mid r_B : \forall ss'tt'. s \Rightarrow s' \to t \Rightarrow t' \to Bst \Rightarrow Bs't' \\ & \mid r_{AB} : \forall tt_1t_2. t \Rightarrow t_1 \to t \Rightarrow t_2 \to At \Rightarrow Bt_1t_2 \end{aligned}$$

Using the second coinduction principle, we show that $\Rightarrow$ is confluent, i.e., if $s \Rightarrow t$ and $s \Rightarrow t'$ then there exists u such that $t \Rightarrow u$ and $t' \Rightarrow u$. The coinductive hypothesis is: for all terms s, t, t', if $s \Rightarrow t$ and $s \Rightarrow t'$ then there exists a red term u^r (i.e., an element of $\texttt{term}^r$) such that $t \Rightarrow^r u^r$ and $t' \Rightarrow^r u^r$.

Let s, t, t' be such that $s \Rightarrow t$ and $s \Rightarrow t'$. We need to show that there exists a green term u^g such that $t \Rightarrow^g u^g$ and $t' \Rightarrow^g u^g$. We do case analysis on the definitions of $s \Rightarrow t$ and $s \Rightarrow t'$. There are the following possibilities.

- $s = t = t' = Ci$. Then take $u^g = C^g i$.
- $s = As_1$ and $t = At_1$ and $t' = At_1'$ with $s_1 \Rightarrow t_1$ and $s_1 \Rightarrow t_1'$. By the coinductive hypothesis we obtain u^r (in $\texttt{term}^r$) such that $t_1 \Rightarrow^r u^r$ and $t_1' \Rightarrow^r u^r$. Take $u^g = A^g u^r$. Then $t = At_1 \Rightarrow^g A^g u^r$ and $t' = At_1' \Rightarrow^g A^g u^r$.
- $s = As_1$ and $t = At_1$ and $t' = Bt_1't_2'$ with $s_1 \Rightarrow t_1$ and $s_1 \Rightarrow t_1'$ and $s_1 \Rightarrow t_2'$. By the coinductive hypothesis we obtain u_1^r, u_2^r such that $t_1 \Rightarrow^r u_1^r$, $t_1' \Rightarrow^r u_1^r$, $t_1 \Rightarrow^r u_2^r$, $t_2' \Rightarrow^r u_2^r$. Take $u^g = B^g u_1^r u_2^r$. Then $s = As_1 \Rightarrow^g B^g u_1^r u_2^r$ and $t' = Bt_1't_2' \Rightarrow^g B^g u_1^r u_2^r$.
- Other cases are analogous to the ones already considered.

The rest of this paper is devoted to precisely stating the two coinduction principles in the logic of Coq, and investigating under which assumptions proofs using these principles may be automatically translated into guarded Coq proofs of the original statement.

3 Formal principles

In this section, we give a precise statement of the two coinduction principles. For this purpose, we define a type system in which our coinductive proofs will be represented. In the next section we define an extension of this type system which will be the target of our translations. Both systems are simplified subsets of the logic of Coq.

The language of our system consists of terms and (co)inductive declarations. First, we present the possible forms of terms together with a brief intuitive explanation of their meaning. The terms are essentially simplified terms of Coq. Below by t, s, u, τ, σ, etc., we denote terms, by c, c', etc., we denote constructors, and by x, y, z, etc., we denote variables. We use $\vec{t}$ for a sequence of terms $t_1 \ldots t_n$ of an unspecified length n, and analogously for a sequence of variables $\vec{x}$. For instance, $s\vec{y}$ stands for $sy_1 \ldots y_n$, where n is not important or implicit in the context. Analogously, we use $\lambda\vec{x} : \vec{\tau}.t$ for $\lambda x_1 : \tau_1.\lambda x_2 : \tau_2. \ldots \lambda x_n : \tau_n.t$, with n implicit or unspecified.

A *term* is a sort $*$, a constructor c, an inductive or a coinductive type I, an application $t_1 t_2$, an abstraction $\lambda x : t_1.t_2$, a dependent product $\forall x : t_1.t_2$, or a case expression $\texttt{case}(t, \lambda\vec{a} : \vec{\alpha}.\lambda x : I\vec{q}\vec{a}.\tau, \lambda\vec{x_1} : \vec{\sigma_1}.s_1 \mid \ldots \mid \lambda\vec{x_k} : \vec{\sigma_k}.s_k)$. In a case expression, t is the term matched on, I is a (co)inductive type, the type of t has the form $I\vec{q}\vec{u}$ where $\vec{q}$ are the values of the parameters, the type $\tau[\vec{u}/\vec{a}, t/x]$ is the return type, i.e., the type of the whole case expression, and $s_i[\vec{v}/\vec{x}]$ is the value of the case expression if the value of t is $c_i\vec{q}\vec{v}$.

For simplicity, we consider only one impredicative sort $*$ of types. If x does not occur free in t_1 then we abbreviate $\forall x : t_1.t_2$ to $t_1 \to t_2$.

A *(co)inductive declaration*

$$I(\vec{p} : \vec{\rho}) : \sigma := c_1 : \sigma_1 \mid \ldots \mid c_n : \sigma_n$$

declares a (co)inductive type I with *parameters* $\vec{p}$ and *arity* $\forall\vec{p} : \vec{\rho}.\sigma$ with n constructors $c_1, \ldots, c_n$ having types $\sigma_1, \ldots, \sigma_n$ respectively. We require:

- $\sigma = \forall\vec{a} : \vec{\alpha}.*$.
- $\sigma_i = \forall x_i^1 : \tau_i^1. \ldots \forall x_i^{k_i} : \tau_i^{k_i}.I\vec{p}\vec{u_i}$.
- I occurs only strictly positively in each σ_i, i.e., I does not occur in $\vec{u_i}$, and for each $j = 1, \ldots, k_i$ either I does not occur in τ_i^j or $\tau_i^j = \forall\vec{y} : \vec{\gamma}.I\vec{s}$ where I does not occur in $\vec{s}$ or $\vec{\gamma}$ ($\vec{s}, \vec{\gamma}$ depend on i, j).

The *arity* of a constructor c_i is $\forall\vec{p} : \vec{\rho}.\sigma_i$, denoted $c_i : \forall\vec{p} : \vec{\rho}.\sigma_i$. For the constructor $c_i : \forall\vec{p} : \vec{\rho}.\forall x_i^1 : \tau_i^1. \ldots \forall x_i^{k_i} : \tau_i^{k_i}.I\vec{p}\vec{u_i}$, the set $\mathcal{R}(c_i)$ of *recursive positions* is the set of all those j for which $\tau_i^j = \forall\vec{y} : \vec{\gamma}.I\vec{s}$.

We have the following reductions:

$$\begin{array}{rcl} (\lambda x : \tau.t)s & \to_\beta & t[s/x] \\ \texttt{case}(c_i\vec{p}\vec{v}, \lambda\vec{a} : \vec{\alpha}.\lambda x : I\vec{p}\vec{a}.\tau, \lambda\vec{x_1} : \vec{\tau_1}.s_1 \mid \ldots \mid \lambda\vec{x_k} : \vec{\tau_k}.s_k) & \to_\iota & s_i[\vec{v}/\vec{x_i}] \end{array}$$

An *environment* is a list of (co)inductive declarations. We write $I \in E$ if a declaration of a (co)inductive type I occurs in the environment E. Analogously, we write $(I : \tau) \in E$ and $(c : \tau) \in E$, if a declaration of I with arity τ occurs in E, or a constructor $c : \tau$ with arity τ in a declaration in E, respectively. A *context* Γ is a list of pairs $x : \tau$ with x a variable and τ a term. A *sort* is $*$ or $\Box$ (note that $\Box$ is not a term, but $*$ is). A *typing judgement* has the form $E; \Gamma \vdash t : \tau$ with t a term and τ a term or a sort. A term t is well-typed and has type τ in the context Γ and environment E if $E; \Gamma \vdash t : \tau$ may be derived using the rules from Figure 1. We denote an empty list by $\langle\rangle$.

In Figure 1 we assume s, s_1, s_2 are sorts. We also assume that the environment E is *well-formed*, which is defined inductively: an empty environment is well-formed, and an environment $E, I(\vec{p} : \vec{\rho}) : \tau := c_1 : \tau_1 \mid \ldots \mid c_n : \tau_n$ (denoted E, I) is well-formed if E is and:

- the constructors $c_1, \ldots, c_n$ are pairwise distinct and distinct from any constructors occurring in the declarations in E;
- $E; \langle\rangle \vdash \forall\vec{p} : \vec{\rho}.\tau : \Box$ and $E; I : \forall\vec{p} : \vec{\rho}.\tau, \vec{p} : \vec{\rho} \vdash \tau_i : *$.

When E, Γ are clear or irrelevant, we write $t : \tau$ instead of $E; \Gamma \vdash t : \tau$.

The type system is a subsystem of the Calculus of Inductive Constructions, so $\beta\iota$-reduction is confluent and strongly normalising on well-typed terms [21]. We usually implictly consider types to be in $\beta\iota$-normal form, without mentioning this every time. An η-expansion changes a term t of type $\forall x : \tau.\sigma$, which is not a λ-abstraction and is such that $x \notin \mathrm{FV}(t)$, into the term $\lambda x : \tau.tx$. The *η-long* form of a term is obtained by η-expanding as much as possible without creating β-redexes.

$$\frac{}{E;\langle\rangle \vdash * : \Box} \qquad \frac{(I:\tau)\in E}{E;\Gamma\vdash I:\tau} \qquad \frac{(c:\tau)\in E}{E;\Gamma\vdash c:\tau}$$

$$\frac{E;\Gamma\vdash A:s \quad x\notin\Gamma}{E;\Gamma,x:A\vdash x:A} \qquad \frac{E;\Gamma\vdash A:B \quad E;\Gamma\vdash C:s \quad x\notin\Gamma}{E;\Gamma,x:C\vdash A:B}$$

$$\frac{E;\Gamma\vdash F:\forall x:A.B \quad E;\Gamma\vdash t:A}{E;\Gamma\vdash Ft:B[t/x]} \qquad \frac{E;\Gamma,x:A\vdash t:B \quad E;\Gamma\vdash(\forall x:A.B):s}{E;\Gamma\vdash(\lambda x:A.t):\forall x:A.B}$$

$$\frac{E;\Gamma\vdash A:s_1 \quad E;\Gamma,x:A\vdash B:s_2}{E;\Gamma\vdash(\forall x:A.B):s_2} \qquad \frac{E;\Gamma\vdash A:B \quad E;\Gamma\vdash B':s \quad B=_{\beta\iota}B'}{E;\Gamma\vdash A:B'}$$

$$\frac{\begin{array}{c}E;\Gamma\vdash t:I\vec{q}\vec{u} \qquad E;\Gamma\vdash(\lambda\vec{a}:\vec{\alpha}.\lambda x:I\vec{q}\vec{a}.\tau):\forall\vec{a}:\vec{\alpha}.I\vec{q}\vec{a}\to *\\ (I(\vec{p}:\vec{\rho}):\forall\vec{a}:\vec{\alpha}.*:=c_1:\forall\vec{x_1}:\vec{\tau_1}.I\vec{p}\vec{u_1}\mid\ldots\mid c_k:\forall\vec{x_k}:\vec{\tau_k}.I\vec{p}\vec{u_k})\in E\\ E;\Gamma\vdash(\lambda\vec{x_i}:\vec{\sigma_i}.s_i):\forall\vec{x_i}:\vec{\sigma_i}.\tau[\vec{w_i}/\vec{a},c_i\vec{q}\vec{x_i}/x] \quad \vec{\sigma_i}=\vec{\tau_i}[\vec{q}/\vec{p}] \quad \vec{w_i}=\vec{u_i}[\vec{q}/\vec{p}]\end{array}}{E;\Gamma\vdash\mathtt{case}(t,\lambda\vec{a}:\vec{\alpha}.\lambda x:I\vec{q}\vec{a}.\tau,\lambda\vec{x_1}:\vec{\sigma_1}.s_1\mid\ldots\mid\lambda\vec{x_k}:\vec{\sigma_k}.s_k):\tau[\vec{u}/\vec{a},t/x]}$$

Figure 1 Typing rules.

A term τ is *first-order* if $*$ does not occur in it. A context Γ is first-order if for every $(x:\tau)\in\Gamma$ the type τ is first-order. A (co)inductive type

$$I(\vec{p}:\vec{\rho}):\sigma:=c_1:\sigma_1\mid\ldots\mid c_n:\sigma_n$$

is first-order if:

- $\sigma,\sigma_1,\ldots,\sigma_n$ are first-order;
- each parameter type ρ_i has the form $\forall\vec{x}:\vec{\tau}.*$ with all $\vec{\tau}$ first-order;
- if $\sigma_i=\forall x_1:\tau_1\ldots\forall x_m:\tau_m.I\vec{p}\vec{u}$ and I occurs in τ_i then $x_i\notin\mathrm{FV}(\tau_{i+1},\ldots,\tau_m,\vec{u})$.

An environment E is first-order if all (co)inductive types in E are. Note that we allow $*$ in the types of parameters of (co)inductive types.

Let $I(\vec{p}:\vec{\rho}):\forall\vec{a}:\vec{\alpha}.*:=c_1:\forall\vec{x_1}:\vec{\tau_1}.I\vec{p}\vec{u_1}\mid\ldots\mid c_k:\forall\vec{x_k}:\vec{\tau_k}.I\vec{p}\vec{u_k}$ be a coinductive declaration. The *red type declaration* $\mathrm{Decl}^r(I)$ for I is

$$I^r:\forall\vec{p}:\vec{\rho}.\forall\vec{a}:\vec{\alpha}.*,\quad \iota_I:\forall\vec{p}:\vec{\rho}.\forall\vec{a}:\vec{\alpha}.I\vec{p}\vec{a}\to I^r\vec{p}\vec{a},\quad \iota_I^g:\forall\vec{p}:\vec{\rho}.\forall\vec{a}:\vec{\alpha}.I^g\vec{p}\vec{a}\to I^r\vec{p}\vec{a}.$$

The *green type declaration* $\mathrm{Decl}^g(I)$ for I is

$$I^g(I^r:\tau_{I^r})(\vec{p}:\vec{\rho}):\forall\vec{a}:\vec{\alpha}.*:=c_1^g:\forall\vec{x_1}:\vec{\tau_1}[I^r/I].I^gI^r\vec{p}\vec{u_1}\mid\ldots\mid c_k^g:\forall\vec{x_k}:\vec{\tau_k}[I^r/I].I^gI^r\vec{p}\vec{u_k}$$

where $\tau_{I^r}=\forall\vec{p}:\vec{\rho}.\forall\vec{a}:\vec{\alpha}.*$ is the arity of the red type I^r. The type I is *admissible for* $E;\Gamma$ if $I^r,\iota_I,\iota_I^g\notin\Gamma$ and $I^g,c_1^g,\ldots,c_k^g\notin E$. Note that I^g need not be first-order, because τ_{I^r} might not have the required form for a parameter type.

We assume two new term forms: $\mathtt{cofix}_1(t)$ and $\mathtt{cofix}_2(t)$.

▶ **Principle 1** (First coinduction principle). *Let $\varphi=\forall\vec{x}:\vec{\tau}.z\vec{u}$ be a first-order type with $z\notin\mathrm{FV}(\vec{\tau},\vec{u})$ free. Let Γ be a first-order context and E a first-order environment. Let $(I:\forall\vec{p}:\vec{\rho}.\forall\vec{a}:\vec{\alpha}.*)\in E$ be a coinductive type admissible for $E;\Gamma$. If*

$$E,\mathrm{Decl}^g(I);\Gamma,\mathrm{Decl}^r(I)\vdash t:\varphi[I^r/z]\to\varphi[I^gI^r/z]$$

then

$$E; \Gamma \vdash \texttt{cofix}_1(t') : \varphi[I/z]$$

where $t' = t[I/I^r, \texttt{id}/\iota_I, \texttt{id}/\iota_I^g, (\lambda I^r.I)/I^g, (\lambda I^r.c_1)/c_1^g, \ldots, (\lambda I^r.c_k)/c_k^g]$ *and* $\texttt{id} = \lambda\vec{p} : \vec{\rho}.\lambda\vec{a} : \vec{\alpha}.\lambda x : I\vec{p}\vec{a}.x$ *and* $c_1, \ldots, c_k$ *are the only constructors of* I*.*

The first coinduction principle could be simply added to our type theory as a typing rule. We conjecture that, even without the first-order restriction, the resulting system would be reasonable and enjoy logical consistency and strong normalisation. In this paper we do not study the meta-theoretical properties of such a system, leaving this for future work. Instead, we investigate to what extent the principle may be implemented in the existing type theory of Coq. It turns out that in addition to the assumptions already stated, we need only minor restrictions on the proof t which have limited practical significance.

▶ Remark 3.1. We believe that the first-order restriction is not necessary for the soundness of the first coinduction principle. It is necessary to enable a translation to guarded Coq proofs. If we allow quantification over types, then some proofs obtained using the principle are not directly translatable, but we believe them to be still valid. For instance, if $I : * := c : I \to I$ and $R : I \to * := r : \forall x : I.Rx \to R(cx)$ are coinductive types and the context contains $F : \forall A : *.A \to A$ then $\texttt{cofix}_1(\lambda f : \forall y.Ry.\lambda y.\texttt{case}(y, \lambda y.Ry, \lambda x.rx(F(Rx)(fx))))$ may be obtained using the first coinduction principle. This proof is not syntactically guarded, but seems valid. Since F is parametric in the type argument A, it cannot inspect its second argument in any way. Hence, the proof is semantically guarded.

In fact, when restricted to streams the first coinduction principle is essentially a degenerate case of the principle from [17] based on sized types. The two colors (red and green) may be seen as two sizes: with green the successor of red. In a proof by coinduction the size needs to increase from red to green. One could extend our principle by introducing an arbitrary number of "colors" corresponding to natural numbers. The resulting system would be very similar to systems based on sized types [3, 2].

The reader may check that the counterexamples to a more relaxed syntactic guardedness criterion from [10, p. 53] do not translate to our principle. Nonetheless, the interaction of the first coinduction principle with impredicative polymorphism and fixpoints is not obvious. We leave for future work the rigorous investigation of the general soundness of our principle without the first-order restriction.

We now proceed to state the second coinduction principle. For this purpose, we need to introduce the definitions of dependent red and green types, as indicated in Section 2. We consider a coinductive type I with the declaration

$$I(\vec{p} : \vec{\rho}) : \forall\vec{a} : \vec{\alpha}.\forall b : J\vec{w}.* := c_1 : \forall\vec{x_1} : \vec{\tau_1}.I\vec{p}\vec{u_1}(d_1\vec{w_1}) \mid \ldots \mid c_k : \forall\vec{x_k} : \vec{\tau_k}.I\vec{p}\vec{u_k}(d_k\vec{w_k})$$

such that each d_i is a constructor of the coinductive type J, and if I occurs in τ_i^j then $\tau_i^j = I\vec{u}v$ and $v = x$ is a variable. We define $\mathrm{Var}_i(I)$ as the set of all variables which appear as the last argument to some occurrence of I in $\vec{\tau_i}$.

A coinductive type I of the above form is *J-admissible* if:

- for every $x \in \mathrm{Var}_i(I)$: x can only occur in $\vec{\tau_i}$ as the last argument of some occurrence of I in $\vec{\tau_i}$, and $x \notin \mathrm{FV}(\vec{u_i})$.
- if $d_i : \forall\vec{y} : \vec{\sigma}.J\vec{t}$ then for every j either $w_i^j = x \in \mathrm{Var}_i(I)$ and $\sigma_j = J\vec{s_j}$, or w_i^j does not contain any variables from $\mathrm{Var}_i(I)$ and σ_j does not contain J.

▶ **Example 3.2.** Let $I : * := c : I \to I$. The type $R : I \to I \to * := r : \forall x, y : I.Rxy \to R(cx)(cy)$ is I-admissible, but $R_1 : I \to I \to * := r_1 : \forall x, y : I.R_1xy \to R_1(cx)y$ and $R_2 : I \to I \to * := r_2 : \forall x, y : I.R_2xy \to R_2(cy)(cy)$ and $R_3 : I \to I \to * := r_3 : \forall x, y : I.R_3yy \to R_3(cx)(cy)$ are not.

The *dependent red type declaration* $\mathrm{Decl}^r_d(I)$ for I is:

$$I^r : \forall \vec{p} : \vec{\rho}.\forall \vec{a} : \vec{\alpha}.\forall b : J^r\vec{w}.*, \quad \iota_I : \forall \vec{p} : \vec{\rho}.\forall \vec{a} : \vec{\alpha}.\forall b : J\vec{w}.I\vec{p}\vec{a}b \to I^r\vec{p}\vec{a}(\iota_J\vec{w}),$$
$$\iota^g_I : \forall \vec{p} : \vec{\rho}.\forall \vec{a} : \vec{\alpha}.\forall b : J^gJ^r\vec{w}.I^gJ^rI^r\vec{p}\vec{a}b \to I^r\vec{p}\vec{a}(\iota^g_J\vec{w}).$$

The *dependent green type declaration* $\mathrm{Decl}^g_d(I)$ for I is:

$$I^g(J^r : \tau_{J^r})(I^r : \tau_{I^r})(\vec{p} : \vec{\rho}) : \forall \vec{a} : \vec{\alpha}.\forall b : J^gJ^r\vec{w}.* :=$$
$$c^g_1 : \forall \vec{x_1} : \vec{\sigma_1}.I^gJ^rI^r\vec{u_1}(d^g_1J^r\vec{w_1}) \mid \ldots \mid c^g_k : \forall \vec{x_k} : \vec{\sigma_k}.I^gJ^rI^r\vec{u_k}(d^g_kJ^r\vec{w_k})$$

where:

- $\sigma^j_i = \tau^j_i[I^r/I]$ if $x^j_i \notin \mathrm{Var}_i(I)$;
- $\sigma^j_i = J^r\vec{v}$ if $x^j_i \in \mathrm{Var}_i(I)$ and $\tau^j_i = J\vec{v}$;
- τ_{J^r} is the arity of the non-dependent red type J^r, and τ_{I^r} is the arity of the dependent red type I^r.

If I is J-admissible then I^g is well-formed.

▶ Remark 3.3. The definition of dependent green types could be relaxed at the cost of additional complexity. For example, we could parameterise the definition by ι_J and allow all constructors of the form $c_i : \forall \vec{x_i} : \vec{\tau_i}.I\vec{p}\vec{u_i}w$ where $I \notin \mathrm{FV}(\vec{\tau_i})$.

▶ **Principle 2** (Second coinduction principle). *Let $\varphi = \forall \vec{x} : \vec{\tau}.\exists y : z\vec{u}.z_1\vec{u_1}y \wedge \ldots \wedge z_n\vec{u_n}y$ be a first-order type with $z, z_1, \ldots, z_n \notin \mathrm{FV}(\vec{\tau}, \vec{u}, \vec{u_1}, \ldots, \vec{u_n})$ free. Let Γ be a first-order context and E a first-order environment. Let $I, I_1, \ldots, I_n \in E$ be coinductive types admissible for $E; \Gamma$, and such that $I_1, \ldots, I_n$ are I-admissible. If*

$$E, \mathrm{Decl}^g(I), \mathrm{Decl}^g_d(I_1), \ldots, \mathrm{Decl}^g_d(I_n); \Gamma, \mathrm{Decl}^r(I), \mathrm{Decl}^r_d(I_1), \ldots, \mathrm{Decl}^r_d(I_n) \vdash$$
$$t : \varphi[I^r/z, I^r_1/z_1, \ldots, I^r_n/z_n] \to \varphi[I^gI^r/z, I^g_1I^r_1I^r_1/z_1, \ldots, I^g_nI^r_nI^r_n/z_n]$$

then $E; \Gamma \vdash \mathtt{cofix}_2(t') : \varphi[I/z, I_1/z_1, \ldots, I_n/z_n]$ where

$$t' = t[I/I^r, \mathtt{id}/\iota_I, \mathtt{id}/\iota^g_I, (\lambda I^r.I)/I^g,$$
$$I_1/I^r_1, \mathtt{id}_1/\iota_{I_1}, \mathtt{id}_1/\iota^g_{I_1}, (\lambda I^rI^r_1.I_1)/I^g_1, \ldots, I_n/I^r_n, \mathtt{id}_n/\iota_{I_n}, \mathtt{id}_n/\iota^g_{I_n}, (\lambda I^rI^r_n.I_n)/I^g_n,$$
$$(\lambda I^r.c_1)/c^g_1, \ldots, (\lambda I^r.c_k)/c^g_k, (\lambda I^rI^r_1.c_{1,1})/c^g_{1,1}, \ldots, (\lambda I^rI^r_1.c_{k,1})/c^g_{k_1,1}, \ldots,$$
$$(\lambda I^rI^r_n.c_{1,n})/c^g_{1,n}, \ldots, (\lambda I^rI^r_n.c_{k_n,n})/c^g_{k_n,n}]$$

and $\mathtt{id}, \mathtt{id}_1, \ldots, \mathtt{id}_n$ are functions of appropriate types which return their last argument, and $c_1, \ldots, c_k$ are the only constructors of I, and $c_{1,j}, \ldots, c_{k_j,j}$ are the only constructors of I_j.

Above we assume all $I^r_1, \ldots, I^r_n$ to be distinct, even if some of the $I_1, \ldots, I_n$ are identical. This weakens the principle slightly in comparison to its informal presentation in Section 2. The types $\exists y : A.B$ and $A \wedge B$ are defined like in Section 2.

As in Section 2, for simplicity we allow only one existential quantifier and we require the existential variable to always be the last argument to a coinductive predicate. The extension to the general case is straightforward but tedious.

4 Proof translations

In this section, we define the two translations which map proofs that use our principles into guarded Coq proofs. The target type system of the translations is the system of the previous section extended with `cofix`.

▶ **Definition 4.1.** We add a new term form to the terms from Section 3: if t is a term and I a coinductive type, then $\texttt{cofix}(\lambda f : \forall \vec{x} : \vec{\tau}.I\vec{u}.t)$ is a term. We extend the type system from Section 3 by the reduction rule

$$\begin{aligned}&\texttt{case}(\texttt{cofix}(\lambda f : \forall \vec{x} : \vec{\tau}.I\vec{u}.t), r, s_1 \mid \ldots \mid s_k) \to_\iota \\ &\qquad \texttt{case}(t[\texttt{cofix}(\lambda f : \forall \vec{x} : \vec{\tau}.I\vec{u}.t)/f], r, s_1 \mid \ldots \mid s_k)\end{aligned}$$

and the typing rule

$$\frac{E; \Gamma \vdash (\lambda f : \forall \vec{x} : \vec{\tau}.I\vec{u}.t) : (\forall \vec{x} : \vec{\tau}.I\vec{u}) \to (\forall \vec{x} : \vec{\tau}.I\vec{u}) \quad \mathcal{G}(f,t)}{E; \Gamma \vdash \texttt{cofix}(\lambda f : \forall \vec{x} : \vec{\tau}.I\vec{u}.t) : \forall \vec{x} : \vec{\tau}.I\vec{u}}$$

where I is a coinductive type, and $\mathcal{G}(f,t)$ states that f is guarded in t, as defined below.

Following [10], we define two predicates $\mathcal{G}_h(f,t)$ for $h = 0, 1$. The predicate $\mathcal{G}_h(f,t)$ holds if one of the following is satisfied:

- $t = \lambda x : \tau.t'$ and $f \notin \mathrm{FV}(\tau)$ and $\mathcal{G}_h(f,t')$;
- $t = \texttt{case}(u, r, s_1 \mid \ldots \mid s_k)w_1 \ldots w_n$ with $n \geq 0$ and $f \notin \mathrm{FV}(u, r, w_1, \ldots, w_n)$ and $\mathcal{G}_h(f, s_i)$ for $i = 1, \ldots, k$;
- $t = ct_1 \ldots t_n$ and for $j = 1, \ldots, n$ we have: either $j \in \mathcal{R}(c)$ is a recusive position and $\mathcal{G}_1(f, t_j)$, or $f \notin \mathrm{FV}(t_j)$;
- $t = f\vec{u}$ and $h = 1$ and $f \notin \mathrm{FV}(\vec{u})$;
- $f \notin \mathrm{FV}(t)$.

We set $\mathcal{G} = \mathcal{G}_0$. If $\mathcal{G}(f,t)$ then f is *guarded in* t.

To avoid confusion, we denote the typability relation in the extended system by $\vdash_e$. We reserve $\vdash$ for the system without `cofix`.

The above syntactic guardedness criterion is more liberal than what is described in [10], but it is closer to the criterion actually implemented in Coq. In [10] terms of the form $\texttt{case}(u, r, s_1 \mid \ldots \mid s_k)w_1 \ldots w_n$ with $n \geq 1$ are not considered. Such terms are often generated by the `destruct` and `inversion` tactics, and Coq's guardedness checker does accept them.

▶ **Example 4.2.** Let $I : * := c : I \to I$. The variable f is guarded in $\texttt{case}(x, \lambda x : I.I \to I, \lambda y : I.\lambda a : I.c(fy))z$ but not in $\texttt{case}(x, \lambda x : I.I \to I, \lambda y : I.\lambda a : I.cy)(fz)$.

▶ **Definition 4.3** (The first translation)**.** Let $\varphi = \forall \vec{x} : \vec{\tau}.z\vec{u}$ be a first-order type with $z \notin \mathrm{FV}(\vec{\tau}, \vec{u})$ free. Let Γ be a first-order context and E a first-order environment. Let

$$I(\vec{p} : \vec{\rho}) : \forall \vec{a} : \vec{\alpha}.* := c_1 : \forall \vec{x_1} : \vec{\tau_1}.I\vec{p}\vec{u_1} \mid \ldots \mid c_k : \forall \vec{x_k} : \vec{\tau_k}.I\vec{p}\vec{u_k}$$

be a coinductive type in E admissible for $E; \Gamma$.

Assume $E, \mathrm{Decl}^g(I); \Gamma, \mathrm{Decl}^r(I) \vdash t : \varphi[I^r/z] \to \varphi[I^g I^r/z]$. The first translation of t, denoted $\mathrm{tr}_1(t)$, is defined as follows:

$$\mathrm{tr}_1(t) = \texttt{cofix}(t'[I/I^r, \texttt{id}/\iota_I, \texttt{id}/\iota_I^g, (\lambda I^r.I)/I^g, (\lambda I^r.c_1)/c_1^g, \ldots, (\lambda I^r.c_k)/c_k^g])$$

where t' is the η-long $\beta\iota$-normal form of t, and $\texttt{id} = \lambda \vec{p} : \vec{\rho}.\lambda \vec{a} : \vec{\alpha}.\lambda x : I\vec{p}\vec{a}.x$.

▶ **Example 4.4.** Let $I : * := c : I \to I$ and $R : I \to * := r : \forall x : I.Rx \to R(cx)$. Then a proof $t = \lambda f : (\forall x : I.R^r x).\lambda x : I.\mathtt{case}(x, \lambda x.R^g x, \lambda x'.r^g x'(fx'))$ for $\forall x : I.Rx$ gets translated to $\mathrm{tr}_1(t) = \mathtt{cofix}(\lambda f : (\forall x : I.Rx).\lambda x : I.\mathtt{case}(x, \lambda x.Rx, \lambda x'.rx'(fx')))$. For readability, we omit parameters to green types.

▶ **Definition 4.5.** A term t satisfies the *proper case restriction for* X, I if for every subterm of t of the form $\mathtt{case}(u, \lambda\vec{a} : \vec{\alpha}.x : J\vec{p}\vec{a}.\tau, s_1 \mid \ldots \mid s_k)$ the type τ is first-order, $J \neq I$, and X, I do not occur in τ. A term t satisfies the *weak case restriction for* X, I if:

- it satisfies the proper case restriction for X, I; or
- $t = \lambda x : \tau.t'$, and t' satisfies the weak case restriction for X, I; or
- $t = \mathtt{case}(u, \lambda\vec{a} : \vec{\alpha}.x : J\vec{p}\vec{a}.\forall\vec{y} : \vec{\beta}.\tau, s_1 \mid \ldots \mid s_k,)\vec{w}$, and $\tau, \vec{\beta}$ are first-order, and X, I do not occur in $\vec{\beta}$, and $J \neq I$, and $s_1, \ldots, s_k$ satisfy the weak case restriction for X, I, and $u, \vec{w}$ satisfy the proper case restriction for X, I.

The proper case restriction allows us to partially recover the subformula property for normal proofs of first-order statements, to the extent that we need it to conclude that the coinductive hypothesis does not occur in a proof of a statement with no occurrences of I^r. This is achieved in the following technical lemma, whose proof may be found in the appendix.

▶ **Lemma 4.6.** *Assume E is a first-order environment, Γ, Γ' is a first-order context, $X, I_X, f_1, \ldots, f_n$ do not occur in Γ, Γ', and u is an η-long $\beta\iota$-normal form satisfying the proper case restriction for X, I_X, and $E, I_X; \Gamma, X : \forall\vec{a} : \vec{\alpha}.*, f_1 : \forall\vec{x} : \vec{\sigma_1}.X\vec{v_1}, \ldots, f_n : \forall\vec{x} : \vec{\sigma_n}.X\vec{v_n}, \Gamma' \vdash u : \tau$.*

1. *If $\tau : *$ is a first-order type and $X, I_X, f_1, \ldots, f_n$ do not occur in τ, then $X, I_X, f_1, \ldots, f_n$ do not occur in u and u is first-order.*
2. *If $\tau = *$ and u is first-order and X, I_X do not occur in u, then $f_1, \ldots, f_n$ do not occur in u.*
3. *If $\tau = I\vec{h}$ and $I \neq I_X$ and either $u = \mathtt{case}(\ldots)\vec{w}$ or $u = y\vec{w}$, then τ is first-order and $X, I_X, f_1, \ldots, f_n$ do not occur in τ.*

▶ **Theorem 4.7.** *Under the assumptions of Definition 4.3, if the η-long $\beta\iota$-normal form of t additionally satisfies the weak case restriction for I^r, I^g, then $E; \Gamma \vdash_e \mathrm{tr}_1(t) : \varphi(I)$.*

Proof. We reason modulo $\beta\iota$-conversion in types. We also implicitly use standard meta-theoretical properties like the generation and subject reduction lemmas [4, 21]. The system is a simplification of the Calculus of Inductive Constructions, and these properties hold.

For any term u, by $\hat{u}$ we denote the η-long $\beta\iota$-normal form of

$$u[I/I^r, \mathtt{id}/\iota_I, \mathtt{id}/\iota_I^g, (\lambda I^r.I)/I^g, (\lambda I^r.c_1)/c_1^g, \ldots, (\lambda I^r.c_k)/c_k^g].$$

Without loss of generality assume t is in η-long $\beta\iota$-normal form. Then $\mathrm{tr}_1(t) = \mathtt{cofix}(\hat{t})$ and $t = \lambda f : \varphi(I^r/z).u$. It follows by induction on the derivation of $E, \mathrm{Decl}^g(I); \Gamma, \mathrm{Decl}^r(I) \vdash t : \varphi[I^r/z] \to \varphi[I^g I^r/z]$ that $E; \Gamma \vdash_e \hat{t} : \varphi[I/z] \to \varphi[I/z]$. Hence, it suffices to show that f is guarded in $\hat{u}$. Recall $\varphi = \forall\vec{x} : \vec{\tau}.z\vec{w}$ with $I^r, I^g, \iota_I, \iota_I^g, f$ not occurring in $\vec{\tau}, \vec{w}$ which are first-order. Because $\hat{u}$ is η-long, $u = \lambda\vec{x} : \vec{\tau}.r$. Hence $E, \mathrm{Decl}^g(I); \Gamma, \mathrm{Decl}^r(I), f : \varphi[I^r/z], \vec{x} : \vec{\tau} \vdash r : I^g I^r\vec{w}$. We need to show that $\mathcal{G}_0(f, \hat{r})$, i.e., f is guarded in $\hat{r}$.

By induction on u in η-long $\beta\iota$-normal form satisfying the weak case restriction for I^r, I^g, we show that if $E'; \Gamma' \vdash u : \sigma$ where $\sigma = I^g I^r\vec{w}'$ (resp. $\sigma = I^r\vec{w}'$), and $E' = E, \mathrm{Decl}^g(I)$, and $\Gamma' = \Gamma, \mathrm{Decl}^r(I), f : \varphi[I^r/z], \vec{x} : \vec{\tau}'$, and $\vec{\tau}', \vec{w}'$ are first-order, and $I^r, I^g, \iota_I, \iota_I^g, f$ do not occur in $\vec{\tau}', \vec{w}'$, then $\mathcal{G}_0(f, \hat{u})$ (resp. $\mathcal{G}_1(f, \hat{u})$).

First, assume $\sigma = I^g I^r\vec{w}'$. We consider possible forms of u.

- $u = xu_1 \dots u_n$. This is impossible, because for no $(x : \tau) \in \Gamma'$ the type τ has I^g or a bound variable at the head of the target.
- $u = cI^r q_1 \dots q_m u_1 \dots u_n$ $(m, n \geq 0)$ where $q_1, \dots, q_m$ are the parameters. Then $c : \forall I^r : \tau_{I^r}.\forall \vec{p} : \vec{\rho}.\forall \vec{x} : \vec{\gamma}.I^g I^r \vec{p} \vec{v}$, and $I^r \notin \mathrm{FV}(\rho)$, and $I^g, \iota_I, \iota_I^g, f$ do not occur in $\vec{\rho}, \vec{\gamma}$, and for each $i = 1, \dots, n$ either $I^r \notin \mathrm{FV}(\gamma_i)$ or $\gamma_i = \forall \vec{y} : \vec{\alpha_i}.I^r \vec{r_i}$ and $i \in \mathcal{R}(c)$ is a recursive position. Since $q_1, \dots, q_m$ occur in $\vec{w}$, $I^r, I^g, \iota_I, \iota_I^g, f$ do not occur in $q_1, \dots, q_m$, and thus $f \notin \mathrm{FV}(\hat{q_1}, \dots, \hat{q_m})$. Also $q_1, \dots, q_m$ are first-order, because $\vec{w}$ is. Let $\gamma_1' = \gamma_1[q_1/p_1] \dots [q_m/p_m]$. Then $I^g, \iota_I, \iota_I^g, f$ do not occur in γ_1', and γ_1' is first-order, and $E'; \Gamma' \vdash u_1 : \gamma_1'$. If $I^r \notin \mathrm{FV}(\gamma_1')$ then $I^r, I^g, \iota_I, \iota_I^g, f$ do not occur in u_1 and u_1 is first-order by Lemma 4.6. So $f \notin \mathrm{FV}(\hat{u_1})$. Otherwise $\gamma_1' = \forall \vec{y} : \vec{\alpha}.I^r \vec{r}$ with $\vec{\alpha}, \vec{r}$ first-order and $I^r, I^g, \iota_I, \iota_I^g, f$ do not occur in $\vec{\alpha}, \vec{r}$, and $1 \in \mathcal{R}(c)$ is a recursive position. Let $\Gamma'' = \Gamma', \vec{y} : \vec{\alpha}$. Because u_1 is η-long, $u_1 = \lambda \vec{y} : \vec{\alpha}.u_1'$ with $E'; \Gamma'' \vdash u_1' : I^r \vec{r}$. By the inductive hypothesis $\mathcal{G}_1(f, \hat{u_1}')$, so $\mathcal{G}_1(f, \hat{u_1})$ because $f \notin \mathrm{FV}(\vec{\alpha})$. Also, x_1 does not occur in $\gamma_2, \dots, \gamma_n$, because I is first-order. Hence, in any case, $\gamma_2' = \gamma_2[q_1/p_1] \dots [q_m/p_m][u_1/x_1]$ is first-order, and $I^g, \iota_I, \iota_I^g, f$ do not occur in γ_2', and $E'; \Gamma' \vdash u_2 : \gamma_2'$. Continuing this argument for $u_2, u_3, \dots, u_n$ we conclude that for each $i = 1, \dots, n$ either $f \notin \mathrm{FV}(\hat{u_i})$, or $\mathcal{G}_1(f, \hat{u_i})$ and $i \in \mathcal{R}(c)$. Since also $f \notin \mathrm{FV}(\hat{q_j})$ for $j = 1, \dots, m$, we conclude that $\mathcal{G}_0(f, \hat{u})$.
- $u = \mathtt{case}(t, r, s_1 \mid \dots \mid s_k) w_1 \dots w_n$ $(n \geq 0)$. Then $E', \Gamma' \vdash t : J \vec{p} \vec{q}$ and either $t = \mathtt{case}(\dots)\vec{h}$ or $t = y\vec{h}$. By the weak case restriction $J \neq I^g$ and t satisfies the proper case restriction. Hence, by Lemma 4.6, $J \vec{p} \vec{q}$ is first-order and $I^r, I^g, \iota_I, \iota_I^g, f$ do not occur in it. Using Lemma 4.6 again, we conclude that $I^r, I^g, \iota_I, \iota_I^g, f$ do not occur in t and t is first-order. Then by the weak case restriction the type $r \vec{q} t =_{\beta\iota} \forall \vec{x} : \vec{\beta}.\zeta$ of $\mathtt{case}(t, r, s_1 \mid \dots \mid s_k)$ must be first-order with I^r, I^g not occurring in $\vec{\beta}$. By point 2 in Lemma 4.6 also ι_I, ι_I^g, f do not occur in $\vec{\beta}$. Now using point 1 of Lemma 4.6 we conclude that $I^r, I^g, \iota_I, \iota_I^g, f$ do not occur in $\vec{w}$ and $\vec{w}$ are first-order, by an argument analogous to the one used in Lemma 4.6 for the case $u = xu_1 \dots u_m$ in the proof of point 1. To sum up, what we have shown so far implies $\mathcal{G}_0(f, \hat{t})$, $\mathcal{G}_0(f, \hat{r})$ and $\mathcal{G}_0(f, \hat{w_i})$ for $i = 1, \dots, n$. It remains to show $\mathcal{G}_0(f, \hat{s_i})$ for $i = 1, \dots, k$. Let the declaration of J be

  $$J(\vec{y} : \vec{\rho}) : \forall \vec{a} : \vec{\alpha}.* := c_1 : \forall \vec{x_1} : \vec{\tau_1}.J \vec{y} \vec{v_1} \mid \dots \mid c_k : \forall \vec{x_k} : \vec{\tau_k}.J \vec{y} \vec{v_k}$$

  We have $E', \Gamma' \vdash s_i : \xi$ where $\xi =_{\beta\iota} \forall \vec{x_i} : \vec{\tau_i}[\vec{p}/\vec{y}].r \vec{v_i}[\vec{p}/\vec{y}](c_i \vec{p} \vec{x_i}) =_{\beta\iota} \forall \vec{x_i} : \vec{\tau_i}[\vec{p}/\vec{y}].\forall \vec{b} : \vec{\beta_1}.I^g I^r \vec{v}$ (see Figure 1). Because $\vec{\tau}, \vec{v_i}, \vec{p}, \vec{\beta_1}$ are first-order and $I^r, I^g, \iota_I, \iota_I^g, f$ do not occur in them, also $\vec{\tau_i}[\vec{p}/\vec{y}], \vec{\beta_1}$ are first-order and $I^r, I^g, \iota_I, \iota_I^g, f$ do not occur in them. Also $\vec{v} = \vec{v_0}[\vec{v_i}[\vec{p}/\vec{y}]/\vec{a}, (c_i \vec{p} \vec{x_i})/x]$. Because $\vec{w}'$ are substitution instances of $\vec{v_0}$, the terms $\vec{v_0}$ must be first-order and $I^r, I^g, \iota_I, \iota_I^g, f$ cannot occur in them. This implies that $\vec{v}$ are first-order and $I^r, I^g, \iota_I, \iota_I^g, f$ do not occur in them. Because s_i is in η-long $\beta\iota$-normal form, $s_i = \lambda \vec{b} : \vec{\beta_1}.s_i'$. Let $\Gamma'' = \Gamma', \vec{x_i} : \vec{\tau_i}[\vec{p}/\vec{y}], \vec{b} : \vec{\beta_1}$. We have $E'; \Gamma'' \vdash s_i' : I^g I^r \vec{v}$, and s_i' still satisfies the weak case restriction, and $\Gamma'', \vec{v}$ satisfy the requirements of the inductive hypothesis. Thus, by the inductive hypothesis we conclude $\mathcal{G}_0(f, \hat{s_i}')$. This shows $\mathcal{G}_0(f, \hat{s_i})$.

Now assume $\sigma = I^r \vec{w}'$. The proof proceeds as above, mutatis mutantdis, except that we have three additional cases.

- $u = fu_1 \dots u_n$. Then u satisfies the proper case restriction. Since $f : \forall \vec{x} : \vec{\tau}.I^r \vec{w}$ with $\vec{\tau}$ first-order and $I^r, I^g, \iota_I, \iota_I^g, f$ not occurring in $\vec{\tau}$, using Lemma 4.6 we may conclude that $f \notin \mathrm{FV}(u_1, \dots, u_n)$ by an inductive argument as in the case $u = xu_1 \dots u_m$ in the proof of Lemma 4.6. Hence $\mathcal{G}_1(f, \hat{u})$.
- $u = \iota_I u_1 \dots u_n$. The argument is then analogous to the case above, because $\iota_I : \forall \vec{p} : \vec{\rho}.\forall \vec{a} : \vec{\alpha}.I \vec{p} \vec{a} \to I^r \vec{p} \vec{a}$ with $\vec{\rho}, \vec{\alpha}$ first-order not containing $I^r, I^g, \iota_I, \iota_I^g, f$.

- $u = \iota^g_I u_1 \dots u_n u'$. We have $\iota^g_I : \forall \vec{p} : \vec{\rho}. \forall \vec{a} : \vec{\alpha}. I^g I^r \vec{p} \vec{a} \to I^r \vec{p} \vec{a}$ with $\vec{\rho}, \vec{\alpha}$ first-order not containing $I^r, I^g, \iota_I, \iota^g_I, f$. Like above, using Lemma 4.6 we conclude that $u_1, \dots, u_n$ are first-order and $I^g, I^r, \iota_I, \iota^g_I, f$ do not occur in them. Also $u' : I^g I^r u_1 \dots u_n$. Hence, by the inductive hypothesis $\mathcal{G}_0(f, \hat{u}')$, so $\mathcal{G}_1(f, \hat{u}')$. Thus $\mathcal{G}_1(f, \hat{u})$, because $\hat{u} = \hat{u}'$. ◀

▶ Remark 4.8. For the purposes of the above theorem, any lemmas used in the proof term must appear in the context Γ. Note that the theorem requires the context and the statement to be first-order, but not the proof term. This implies that if the statement is first-order and we recursively unfold the proofs of all lemmas, we obtain a proof term t which satisfies the requirements of the theorem in the empty context, even if the lemmas used in the proof term were not first-order; provided the weak case restriction holds for the η-long $\beta\iota$-normal form of t.

One important situation where this procedure fails is when using the setoid library for rewriting. Then the generated proof terms, after unfolding, often fail to satisfy the weak case restriction.

To ease working with equality on coinductive types, our plugin provides a `peek` tactic which forces reduction of a cofixpoint.

The idea with the second translation is to "split" a coinductive proof $t : \forall \vec{x} : \vec{\tau}. \exists y : I \vec{u}. I_1 \vec{u_1} y \wedge \dots \wedge I_n \vec{u_n} y$ into $n+1$ separate guarded proofs $t_0 : \forall \vec{x} : \vec{\tau}. I \vec{u}$ and $t_i : \forall \vec{x} : \vec{\tau}. I_i \vec{u_i}(t_0 \vec{x})$ for $i = 1, \dots, n$.

▶ **Definition 4.9** (The second translation)**.** Let $\varphi = \forall \vec{x} : \vec{\tau}. \exists y : z \vec{u}. z_1 \vec{u_1} y \wedge \dots \wedge z_n \vec{u_n} y$ be a first-order type with $z, z_1, \dots, z_n \notin \mathrm{FV}(\vec{\tau}, \vec{u}, \vec{u_1}, \dots, \vec{u_n})$ free. Let Γ be a first-order context and E a first-order environment. Let $I, I_1, \dots, I_n \in E$ be coinductive types admissible for $E; \Gamma$, and such that $I_1, \dots, I_n$ are I-admissible. Assume

$$E, \mathrm{Decl}^g(I), \mathrm{Decl}^g_d(I_1), \dots, \mathrm{Decl}^g_d(I_n); \Gamma, \mathrm{Decl}^r(I), \mathrm{Decl}^r_d(I_1), \dots, \mathrm{Decl}^r_d(I_n) \vdash$$
$$t : \varphi[I^r/z, I^r_1/z_1, \dots, I^r_n/z_n] \to \varphi[I^g I^r/z, I^g_1 I^r I^r_1/z_1, \dots, I^g_n I^r I^r_n/z_n].$$

The second translation of t, denoted $\mathrm{tr}_2(t)$, is defined as follows. We omit the parameters of `ex_intro` and `conj`.

1. We compute t' – the η-long $\beta\iota$-normal form of

$$\lambda f : \psi[I^r/z]. \lambda f_1 : \psi_1[I^r_1/z, f/y] \dots \lambda f_n : \psi_n[I^r_n/z, f/y].$$
$$t(\lambda \vec{x} : \vec{\tau}. \texttt{ex\_intro}(f \vec{x})(\texttt{conj}(f_1 \vec{x})(f_2 \vec{x})(\dots))).$$

where $\psi = \forall \vec{x} : \vec{\tau}. z \vec{u}$ and $\psi_i = \forall \vec{x} : \vec{\tau}. z \vec{u_i}(y \vec{x})$. In this way we "split" the coinductive hypothesis into $n+1$ hypotheses. Note that

$$t' : \forall f : \psi[I^r/z]. \psi_1[I^r_1/z, f/y] \to \dots \to \psi_n[I^r_n/z, f/y] \to$$
$$\varphi[I^g I^r/z, I^g_1 I^r I^r_1/z_1, \dots, I^g_n I^r I^r_n/z_n].$$

2. Inductively, for a term u we define $\mathrm{tr}^0_2(u)$, and for $i = 1, \dots, n$, a sequence of terms $\vec{w}$, and a term u, we define $\mathrm{tr}^i_2(\vec{w}; u)$.
   - If $u = \texttt{case}(x, \lambda x : J \vec{p}. \zeta, \lambda \vec{x_1} : \vec{\tau_1}. s_1 \mid \dots \mid \lambda \vec{x_k} : \vec{\tau_k}. s_k)$ where

$$\zeta = \exists y : I^g I^r \vec{v}. I^g_1 I^r I^r_1 \vec{v_1} y \wedge \dots \wedge I^g_n I^r I^r_n \vec{v_n} y$$

   and J is a (co)inductive type with no non-parameter arguments, and $c_1, \dots, c_k$ are the only constructors of J, then
   - $\mathrm{tr}^0_2(u) = \texttt{case}(x, \lambda x : J \vec{p}. I \vec{v}, \lambda \vec{x_1} : \vec{\tau_1}. \mathrm{tr}^0_2(s_1) \mid \dots \mid \lambda \vec{x_k} : \vec{\tau_k}. \mathrm{tr}^0_2(s_k))$.
   - $\mathrm{tr}^i_2(\vec{w}; u) = \texttt{case}(x, \lambda x : J \vec{p}. I_i \vec{v_i}(f \vec{w}), \lambda \vec{x_1} : \vec{\tau_1}. \mathrm{tr}^i_2(\vec{w}[c_1 \vec{p} \vec{x_1}/x]; s_1) \mid \dots \mid \lambda \vec{x_k} : \vec{\tau_k}. \mathrm{tr}^i_2(\vec{w}[c_k \vec{p} \vec{x_k}/x]; s_k))$ for $i > 0$.

- If $u = \texttt{ex\_intro}\, u_0\, (\texttt{conj}\, u_1\, (\texttt{conj}\, u_2\, (\ldots)))$ then
  - $\mathrm{tr}_2^0(u) = u_0[I/I^r, \texttt{id}/\iota_I, (\lambda I^r.I)/I^g, \ldots]$,
  - $\mathrm{tr}_2^i(\vec{w}; u) = u_i[I/I^r, \texttt{id}/\iota_I, (\lambda I^r.I)/I^g, \ldots]$,

  with the substitutions like in Principle 2.

  In other cases tr_2^i are undefined.
3. Since t' is in η-long $\beta\iota$-normal form,

$$t' = \lambda f : \psi[I^r/z].\lambda f_1 : \psi_1[I_1^r/z, f/y] \ldots \lambda f_n : \psi_n[I_n^r/z, f/y].\lambda \vec{x} : \vec{\tau}.t''.$$

   We define t_i for $i = 0, \ldots, n$ by:
   - $t_0 = \texttt{cofix}(\lambda f : \psi[I/z].\lambda \vec{x} : \vec{\tau}.\mathrm{tr}_2^0(t''))$,
   - $t_i = \texttt{cofix}(\lambda f_i : \psi_i[I_i/z, t_0/y].\lambda \vec{x} : \vec{\tau}.\mathrm{tr}_2^i(\vec{x}; t'')[t_0/f])$ for $i = 1, \ldots, n$.
4. Finally: $\mathrm{tr}_2(t) = \lambda \vec{x} : \vec{\tau}.\texttt{ex\_intro}(t_0\vec{x})(\texttt{conj}(t_1\vec{x})(\texttt{conj}(t_2\vec{x})(\ldots(\texttt{conj}(t_{n-1}\vec{x})(t_n\vec{x})))))$.

▶ Remark 4.10. The above translation is very restricted, essentially to proofs which do case analysis on variables followed by the use of the coinductive hypothesis. It is undefined for proof terms commonly generated by the `inversion` tactic. The problem here is that Coq's dependent matching "forgets" some equality information in the branches, which makes it difficult to automatically choose the arguments for the f function above in a way that satisfies the type-checker. More precisely, when $u : I\vec{q}\vec{w}$ and $c_i : \forall \vec{p} : \vec{\rho}.\forall \vec{x_i} : \vec{\tau_i}.I\vec{p}\vec{v_i}$ is the i-th constructor of I, the equalities $u = c_i\vec{p}\vec{v_i}$ and $v_i^j = w_i^j$ are not available to the Coq's type-checking algorithm when checking the branch s_i in $\texttt{case}(u, r, s_1 \mid \ldots \mid s_k)$ (see Figure 1). In practice, we allow a broader class of proof terms. In particular, we handle proofs commonly generated by one inversion (but not multiple nested inversions in general). The details of this are very tedious and not particularly illuminating, so we do not describe them here. The restrictions in the actual implementation, while a bit ad-hoc and still significant, are weak enough to allow for reasonably convenient usage. Especially the restriction on nested inversions can usually be easily worked around. See Example 5.2.

▶ **Example 4.11.** Let $I : * := c : I \to I$ and $R : I \to I \to * := r : \forall x, y : I.Rxy \to R(cx)(cy)$ be coinductive types. Recall $I^r : *$, $R^r : I \to I^r \to *$, $I^g : * := c^g : I^r \to I^g$, $R^g : I \to I^g \to * := r^g : \forall x : I.\forall y : I^r.R^r xy \to R^g(cx)(c^g y)$. For readability, we omit the parameters to the green types. Consider the term

$$\begin{aligned} t \quad = \quad & \lambda f : (\forall x : I.\exists y : I^r.R^r xy).\lambda x : I.\texttt{case}(x, \lambda x.\exists y : I^g.R^g xy, \\ & \lambda x'.\texttt{case}(fx', \exists y : I^g.R^g(cx')y, \lambda y' : I^r.\lambda h : R^r x'y'.\texttt{ex\_intro}(c^g y')(r^g x'y'h))) \end{aligned}$$

which proves $\forall x : I.\exists y : I.Rxy$ by the second coinduction principle. After the first step of the second translation we obtain $t' = \lambda f : I \to I^r.\lambda f_1 : \forall x : I.R^r x(fx).\lambda x : I.t''$ where $t'' = \texttt{case}(x, \lambda x.\exists y : I^g.R^g xy, \lambda x'.\texttt{ex\_intro}(c^g(fx'))(r^g x'(fx')(f_1 x')))$. We have $\mathrm{tr}_2^0(t'') = \texttt{case}(x, \lambda x.I, \lambda x'.c(fx'))$ and $\mathrm{tr}_2^1(x; t'') = \texttt{case}(x, \lambda x.Rx(fx), \lambda x'.rx'(fx')(f_1 x'))$. Then $\mathrm{tr}_2(t) = \lambda x{:}I.\texttt{ex\_intro}(t_0 x)(t_1 x)$ with $t_0 = \texttt{cofix}\lambda f : I \to I.\lambda x{:}I.\mathrm{tr}_2^0(t'')$ and $t_1 = \texttt{cofix}\lambda f_1 : (\forall x : I.Rx(t_0 x)).\lambda x : I.\mathrm{tr}_2^1(x; t'')[t_0/f]$.

Note that in the example above $rx'(t_0 x')(f_1 x') : R(cx')(c(t_0 x'))$, but the branch should have type $R(cx')(t_0(cx'))$. We thus relax the ι-reduction for `cofix` to: $\texttt{cofix}(\lambda f.t) \to_\iota t[\texttt{cofix}(\lambda f.t)/f]$. Then $t_0(cx') =_{\beta\iota} c(t_0 x')$ and $\mathrm{tr}_2(t)$ type-checks. The typing relation of the system thus modified is denoted by $\vdash_{e+}$. This allows us to prove the theorem below, but makes type-checking undecidable. In practice, the implemented translation inserts appropriate equality proofs into the proof term to make it type-check. The details are again quite tedious and not very interesting.

▶ **Definition 4.12.** A term satisfies the *strong case restriction* if it does not contain any subterms $\mathtt{case}(\mathtt{case}(u, r', t_1 \mid \ldots \mid t_m)\vec{w}, r, s_1 \mid \ldots \mid s_k)$ or $\mathtt{case}(u, r, s_1 \mid \ldots \mid s_k)w_1 \ldots w_n$ with $n \geq 1$.

▶ **Theorem 4.13.** *Under the assumptions of Definition 4.9, if the η-long $\beta\iota$-normal form of t additionally satisfies the strong case restriction, and $\mathrm{tr}_2(t)$ is defined, and $E; \Gamma, f : \psi[I/z], f_i : \psi_i[I_i/z, t_0/y], \vec{x} : \vec{\tau} \vdash_{e+} \mathrm{tr}_2^i(\vec{x}; t'')[t_0/f] : I_i\vec{u_i}(t_0\vec{x})$ for $i = 1, \ldots, n$ (with $\psi, \psi_i, t_0, \ldots$ as in Definition 4.9), then $E; \Gamma \vdash_{e+} \mathrm{tr}_2(t) : \varphi(I; I_1, \ldots, I_n)$.*

Proof (sketch). The strong case restriction essentially recovers the subformula property for first-order statements, which allows us to show that the coinductive hypotheses do not appear in parts of the proof term whose types do not contain corresponding red or green types. The special form of the original proof term enforced by the second translation, and the typability assumptions for tr_2^i, guarantee that the result is well-typed. ◀

5 Coq plugin

We provide a proof-of-concept implementation of our principles in a Coq plugin (see the supplement material). The plugin introduces the `CoInduction` command which starts a proof by coinduction using one of our principles. The command defines the (dependent) green coinductive types, and adds the (dependent) red type declarations and the coinductive hypothesis to the context. At `Qed` the proof is translated to a guarded Coq proof. The coinduction principle is chosen automatically based on the form of the goal statement.

▶ **Example 5.1.** Using our plugin, the proofs from Example 2.2 may be formalised as follows.

```
CoInduction lem_refl : forall {A : Type} (s : Stream A), s == s.
Proof. ccrush. Qed.

CoInduction lem_sym :
   forall {A : Type} (s1 s2 : Stream A), s1 == s2 -> s2 == s1.
Proof. ccrush. Qed.

CoInduction lem_trans :
   forall {A : Type} (s1 s2 s3 : Stream A), s1 == s2 -> s2 == s3 -> s1 == s3.
Proof. destruct 1; ccrush. Qed.
```

The `ccrush` tactic is a generic proof search tactic, based on a tactic from CoqHammer [8]. Note that the user may just apply generic automated tactics without worrying too much about the guarded use of the coinductive hypothesis. In contrast, when using Coq's `cofix` directly, generic automated tactics are likely to use the coinductive hypothesis incorrectly so that the proof fails at `Qed`.

▶ **Example 5.2.** A direct formalisation of the confluence proof from Example 2.4 would require two nested inversions, and the second translation would fail at `Qed` (compare Remark 4.10). This may, however, be easily worked around by defining a predicate $\mathtt{Peak}\, s\, t\, t'$ which holds if $s \Rightarrow t$ and $s \Rightarrow t'$ do. We define all predicates into `Set` to get around Coq's restriction of case analysis on proofs. Note that for the proof of `lem_peak` below the first translation may be used, with no restrictions on nested destructions/inversions.

```
CoInductive Peak : term -> term -> term -> Set :=
| peak_C : forall i, Peak (C i) (C i) (C i)
| peak_A : forall s t t', Peak s t t' -> Peak (A s) (A t) (A t')
| peak_B : forall s t s1 t1 s2 t2, Peak s s1 s2 -> Peak t t1 t2 ->
                                   Peak (B s t) (B s1 t1) (B s2 t2)
| peak_AAB : forall s s' t1 t2, Peak s s' t1 -> Peak s s' t2 ->
                                Peak (A s) (A s') (B t1 t2)
| peak_ABA : forall s s' t1 t2, Peak s t1 s' -> Peak s t2 s' ->
                                Peak (A s) (B t1 t2) (A s')
| peak_ABB : forall s s1 s2 t1 t2, Peak s s1 t1 -> Peak s s2 t2 ->
                                   Peak (A s) (B s1 s2) (B t1 t2).

CoInduction lem_peak : forall s t t', s ==> t -> s ==> t' -> Peak s t t'.
Proof. destruct 1; inversion_clear 1; constructor; eauto. Qed.
```

Then the confluence proof looks as follows. It corresponds closely to the "pen-and-paper" proof presented in Example 2.4.

```
CoInduction lem_confl :
   forall s t t', Peak s t t' -> { s' & (t ==> s') * (t' ==> s') }.
Proof. intros s t t' H; inversion_clear H.
  - ccrush.
  - generalize (CH s0 t0 t'0 H0); intro.
    simp_hyps; eexists; split; constructor; eauto.
  - generalize (CH s0 s1 s2 H0); generalize (CH t0 t1 t2 H1); intros.
    simp_hyps; eexists; split; constructor; eauto.
  (...)
Qed.
```

Above `CH` refers to the coinductive hypothesis automatically introduced by `CoInduction`:

```
CH : forall s t t' : term, Peak s t t' ->
       {s' : term__r & Red__r__01 t s' * Red__r__00 t' s'}
```

At the beginning of the proof the goal is:

```
forall s t t' : term, Peak s t t' ->
   {s' : term__g term__r & Red__g__01 term__r Red__r__01 t s' *
                           Red__g__01 term__r Red__r__00 t' s'}
```

▶ **Example 5.3.** Using the first coinduction principle, we formalised most of the examples from [14]. The formalisation is in the `examples/practical.v` file in the plugin sources.

6 Conclusions and future work

We introduced two coinduction principles and corresponding proof translations which, under certain conditions, map proofs using our principles to guarded Coq proofs. In contrast to previous work on coinduction, the second principle allows to directly prove by coinduction statements with existential quantifiers and multiple coinductive predicates in the conclusion. The proof translations clarify the relationship between Coq's syntactic guardedness criterion and the shape of normal forms of proofs obtained using our principles. Implementating the first translation required only small restrictions on dependent matches occurring in proof

terms. While trying to implement the second translation, we encountered more difficulties, which necessitated introducing much heavier restrictions. These difficulties, however, do not seem to be fundamental, but rather stem from the limitations of Coq's type theory.

The restrictions on proof terms are needed because normal proofs in the Calculus of Inductive Constructions are not sufficiently normal in a proof-theoretical sense. And they cannot be normalised further using commutative conversions [20, Chapter 6], like in a natural-deduction system for first-order logic, because commutative conversions are not sound in general for dependent elimination with `case` as defined in the Calculus of Inductive Constructions. The lack of "good" normal forms is a consequence of the fact that not enough equality information is available to the type checker in the branches of dependent matches, which also makes it difficult to implement the second coinduction principle in full generality.

For a more complete implementation of the second coinduction principle, it would probably be better to use a target system with copatterns and sized types, or use negative coinductive types instead of positive ones. We leave this for future work. It also remains to investigate the properties of a type theory directly extended with our principles, and the effects of removing the first-order restriction.

References

1. A. Abel. *A Polymorphic Lambda-Calculus with Sized Higher-Order Types*. PhD thesis, Ludwig-Maximilians-Universität München, 2006.
2. A. Abel and B. Pientka. Well-founded recursion with copatterns and sized types. *J. Funct. Program.*, 26:e2, 2016.
3. A. Abel, A. Vezzosi, and T. Winterhalter. Normalization by evaluation for sized dependent types. *PACMPL*, 1(ICFP):33:1–33:30, 2017.
4. H. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science*, volume 2, pages 118–310. Oxford University Press, 1992.
5. T. Coquand. Infinite Objects in Type Theory. In *TYPES 1993*, pages 62–78, 1993.
6. Ł. Czajka. A New Coinductive Confluence Proof for Infinitary Lambda Calculus. Submitted, 2018.
7. Ł. Czajka. Coinduction: an elementary approach. *arXiv*, 2019. `arXiv:1501.04354`.
8. Ł. Czajka and C. Kaliszyk. Hammer for Coq: Automation for Dependent Type Theory. *J. Autom. Reasoning*, 61(1-4):423–453, 2018.
9. H. Geuvers. Inductive and coinductive types with iteration and recursion. In *TYPES 1992*, pages 193–217, 1992.
10. E. Giménez. Codifying Guarded Definitions with Recursive Schemes. In *TYPES*, pages 39–59, 1994.
11. J. Hughes, L. Pareto, and A. Sabry. Proving the Correctness of Reactive Systems Using Sized Types. In *POPL 1996*, pages 410–423, 1996.
12. C.-K. Hur, G. Neis, D. Dreyer, and V. Vafeiadis. The power of parameterization in coinductive proof. In *POPL 2013*, pages 193–206, 2013.
13. B. Jacobs and J. Rutten. An introduction to (co)algebras and (co)induction. In *Advanced Topics in Bisimulation and Coinduction*, pages 38–99. Cambridge University Press, 2011.
14. D. Kozen and A. Silva. Practical coinduction. *Math. Struct. in Comp. Science*, 27(7):1132–1152, 2017.
15. N.P. Mendler. Inductive types and type constraints in the second-order lambda calculus. *Annals of Pure and Applied Logic*, 51:159–172, 1991.
16. J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.
17. J.L. Sacchini. Type-Based Productivity of Stream Definitions in the Calculus of Constructions. In *LICS 2013*, pages 233–242, 2013.

18 D. Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

19 A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

20 A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 1996.

21 B. Werner. *Une Théorie des Constructions Inductives*. PhD thesis, Paris Diderot University, France, 1994.