First-order guarded coinduction in Coq

Łukasz Czajka, TU Dortmund University

September 2019
Coinduction

A method to define and reason about potentially infinite objects.
Coinduction

A method to define and reason about potentially infinite objects.

CoInductive Stream (A : Type) : Type :=
| cons : A -> Stream A -> Stream A.

CoInductive EqSt {A : Type} : Stream A -> Stream A -> Prop :=
| eqst : forall x s1 s2, EqSt s1 s2 ->
               EqSt (cons x s1) (cons x s2).

Notation "A ≈ B" := (EqSt A B) (at level 70).
Lemma lem_refl : forall {A : Type} (s : Stream A), s ≈ s.
Proof.
  cofix CH.
  eauto.
Lemma lem_refl : \forall \{A : Type\} (s : Stream A), s \approx s.
Proof.
  cofix CH.
  eauto.

No more subgoals.
Lemma lem_refl : forall {A : Type} (s : Stream A), s ≈ s.
Proof.
  cofix CH.
eauto.
Qed.
Lemma lem_refl : forall {A : Type} (s : Stream A), s ≈ s.
Proof.
  cofix CH.
  eauto.
Qed.

Error:
Recursive definition of CH is ill-formed.
In environment
CH : forall (A : Type) (s : Stream A), s == s
Unguarded recursive call in "CH".
Recursive definition is: "CH".
Lemma lem_refl : forall {A : Type} (s : Stream A), s ≈ s.
Proof.
  cofix CH.
  destruct s.
  eauto.
Qed.
Coinduction in Coq

Lemma lem_refl : forall {A : Type} (s : Stream A), s ≈ s.
Proof.
  cofix CH.
  destruct s.
  eauto.
Qed.

Error:
Recursive definition of CH is ill-formed.
In environment
CH : forall (A : Type) (s : Stream A), s == s
A : Type
s : Stream A
a : A
s0 : Stream A
Unguarded recursive call in "CH A (cons a s0)".
Recursive definition is:
"fun (A : Type) (s : Stream A) => match s as s0 return (s0 == s0 end | cons a s0 => CH A (cons a s0) end".
Lemma lem_refl : forall {A : Type} (s : Stream A), s ≈ s.
Proof.
  cofix CH.
  destruct s.
  constructor.
  eauto.
Qed.
Lemma lem_refl : forall {A : Type} (s : Stream A), s ≈ s.
Proof.
  cofix CH.
  destruct s.
  constructor.
  eauto.
Qed.

Finally works!
Coinduction in Coq

Lemma lem_refl : forall {A : Type} (s : Stream A), s ≈ s.
Proof.
  cofix CH.
  destruct s.
  constructor.
  eauto.
Qed.

Finally works!

But this is just a very simple example...
CoInduction lem_refl :
  forall {A : Type} (s : Stream A), s ≈ s.
Proof.
  cc crush.
Qed.
A coinduction principle for Coq

- Ensures guarded use of the coinductive hypothesis.
A coinduction principle for Coq

· Ensures guarded use of the coinductive hypothesis.
  · Interacts well with generic automated tactics.
A coinduction principle for Coq

- Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.
    - Paco: Coq library for parametric coinduction.
A coinduction principle for Coq

- Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.
    - Paco: Coq library for parametric coinduction.
- Implementable in the existing type theory of Coq, via a proof translation to guarded Coq proofs (under certain assumptions).
A coinduction principle for Coq

- Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.
    - Paco: Coq library for parametric coinduction.
- Implementable in the existing type theory of Coq, via a proof translation to guarded Coq proofs (under certain assumptions).
  - No reformulation of existing definitions or proofs necessary.
A coinduction principle for Coq

- Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.
    - Paco: Coq library for parametric coinduction.
- Implementable in the existing type theory of Coq, via a proof translation to guarded Coq proofs (under certain assumptions).
  - No reformulation of existing definitions or proofs necessary.
  - A new CoInduction command starts a proof by coinduction using our principle.
A coinduction principle for Coq

- Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.
    - Paco: Coq library for parametric coinduction.
- Implementable in the existing type theory of Coq, via a proof translation to guarded Coq proofs (under certain assumptions).
  - No reformulation of existing definitions or proofs necessary.
  - A new CoInduction command starts a proof by coinduction using our principle.
- Corresponds closely to informal “pen-and-paper” proofs by coinduction.
A coinduction principle for Coq

- Corresponds closely to informal “pen-and-paper” proofs by coinduction.
A coinduction principle for Coq

- Corresponds closely to informal “pen-and-paper” proofs by coinduction.
  - Silva, Kozen, “Practical coinduction”, MSCS 2017

Lemma
\( \approx \) is reflexive.

Proof.
Let \( s \) be a stream. We have \( s = \text{cons} \ s' \). By the coinductive hypothesis \( s' \approx s' \). Hence \( \text{cons} \ x \ s' \approx \text{cons} \ x \ s' \) by the definition of \( \approx \). \( \square \)
Lemma
≈ is reflexive.

Proof.
Let $s$ be a stream. We have $s = \text{cons } x \ s'$. By the coinductive hypothesis $s' \approx s'$. Hence $\text{cons } x \ s' \approx \text{cons } x \ s'$ by the definition of $\approx$. 

\qed
Lemma
\( \approx \) is reflexive.

Proof.
Let \( s \) be a stream. We have \( s = \text{cons} \, x \, s' \). By the coinductive hypothesis \( s' \approx^r s' \). Hence \( \text{cons} \, x \, s' \approx^g \text{cons} \, x \, s' \) by the definition of \( \approx^g \). \( \square \)
An informal coinductive proof

Lemma
If $\approx^r$ is reflexive then $\approx^g$ is reflexive.

Proof.
Let $s$ be a stream. We have $s = \text{cons } x \ s'$. By the coinductive hypothesis $s' \approx^r s'$. Hence $\text{cons } x \ s' \approx^g \text{cons } x \ s'$ by the definition of $\approx^g$. 

\qed
Red and green types

For each coinductive type $I : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.\ast$ we need to define two associated types: the red type $I^r : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.\ast$ and the green type $I^g : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.\ast$. 

$I^r$ is the type of red values (proofs) obtained from the coinductive hypothesis. Ensures guarded use of the coinductive hypothesis: prohibits case analysis on red values or using red values with functions/lemmas expecting values of type $I^r$.

$I^g$ is the type of green values (proofs) that need to be produced in the conclusion. Ensures productivity: to obtain a green value from a red value a constructor must be applied.
Red and green types

For each coinductive type $I : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.\ast$ we need to define two associated types: the red type $I^r : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.\ast$ and the green type $I^g : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.\ast$.

- $I^r$ is the type of red values (proofs) obtained from the coinductive hypothesis.
Red and green types

For each coinductive type $I : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.*$ we need to define two associated types: the red type $I^r : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.*$ and the green type $I^g : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.*$.

- $I^r$ is the type of red values (proofs) obtained from the coinductive hypothesis.
  - Ensures guarded use of the coinductive hypothesis: prohibits case analysis on red values or using red values with functions/lemmas expecting values of type $I$.
Red and green types

For each coinductive type $I : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.\ast$ we need to define two associated types: the red type $I^r : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.\ast$ and the green type $I^g : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k.\ast$.

- $I^r$ is the type of red values (proofs) obtained from the coinductive hypothesis.
  - Ensures guarded use of the coinductive hypothesis: prohibits case analysis on red values or using red values with functions/lemmas expecting values of type $I$.
- $I^g$ is the type of green values (proofs) that need to be produced in the conclusion.
Red and green types

For each coinductive type \( I : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k \) we need to define two associated types: the red type \( I^r : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k \) and the green type \( I^g : \Pi x_1 : \sigma_1 \ldots \Pi x_k : \sigma_k \).

- \( I^r \) is the type of red values (proofs) obtained from the coinductive hypothesis.
  - Ensures guarded use of the coinductive hypothesis: prohibits case analysis on red values or using red values with functions/lemmas expecting values of type \( I \).
- \( I^g \) is the type of green values (proofs) that need to be produced in the conclusion.
  - Ensures productivity: to obtain a green value from a red value a constructor must be applied.
Red types

- $I^r$ is a fresh type symbol.
Red types

- $I^r$ is a fresh type symbol.
- Any value in $I s_1 \ldots s_k$ or in $I^g s_1 \ldots s_k$ may be converted into the corresponding value in $I^r s_1 \ldots s_k$. 
Red types

- $I^r$ is a fresh type symbol.
- Any value in $I_{s_1 \ldots s_k}$ or in $I^g_{s_1 \ldots s_k}$ may be converted into the corresponding value in $I^r_{s_1 \ldots s_k}$.
  - But it cannot be converted back!
Red types

- $I^r$ is a fresh type symbol.
- Any value in $I s_1 \ldots s_k$ or in $I^g s_1 \ldots s_k$ may be converted into the corresponding value in $I^r s_1 \ldots s_k$.
  - But it cannot be converted back!
  - It can be converted to a “larger” green value by applying a constructor.
Green types

The green type $I^g$ is an inductive type such that for every constructor

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

of $I$ there is a corresponding green constructor

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

- For the type of streams \( \text{Stream} \) the green type \( \text{Stream}^g \) is:

\[
\text{Stream}^g(A : \ast) : \ast := \text{cons}^g : A \rightarrow \text{Stream}^r A \rightarrow \text{Stream}^g A
\]
Green types

· For the type of streams $\text{Stream}$ the green type $\text{Stream}^g$ is:

$$\text{Stream}^g(A : \ast) : \ast := \text{cons}^g : A \rightarrow \text{Stream}^r A \rightarrow \text{Stream}^g A$$

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

$$\text{EqSt}^g(A : \ast) : \text{Stream} A \rightarrow \text{Stream} A \rightarrow \ast :=
\text{eqst}^g : \forall x : A.\forall s_1, s_2 : \text{Stream} A.
\text{EqSt}^r A s_1 s_2 \rightarrow \text{EqSt}^g A (\text{cons} x s_1) (\text{cons} x s_2)$$
First coinduction principle

For $\varphi = \forall x_1 : \tau_1 \ldots \forall x_n : \tau_n. I s_1 \ldots s_k$ we write $\varphi(I') = \forall x_1 : \tau_1 \ldots \forall x_n : \tau_n. I' s_1 \ldots s_k$. 
First coinduction principle

For $\varphi = \forall x_1 : \tau_1 \ldots \forall x_n : \tau_n. I s_1 \ldots s_k$ we write $\varphi(I') = \forall x_1 : \tau_1 \ldots \forall x_n : \tau_n. I' s_1 \ldots s_k$.

Principle (First coinduction principle – informal)

Let $I$ be a coinductive type and $\varphi(I)$ a first-order statement. If $\varphi(I')$ implies $\varphi(I^g)$ then $\varphi(I)$ holds.
First coinduction principle

Let

\[ I(\vec{p} : \vec{\rho}) : \forall \vec{a} : \vec{\alpha}. \ast := \]
\[ c_1 : \forall \vec{x}_1 : \vec{\tau}_1. I\vec{p}\vec{u}_1 | \ldots | c_k : \forall \vec{x}_k : \vec{\tau}_k. I\vec{p}\vec{u}_k \]

be a coinductive declaration.
First coinduction principle

· Let

\[ I(\vec{p} : \vec{\rho}) : \forall \vec{a} : \vec{\alpha}. \ast := \]
\[ c_1 : \forall \vec{x}_1 : \vec{\tau}_1. I\vec{p}_1 \mid \ldots \mid c_k : \forall \vec{x}_k : \vec{\tau}_k. I\vec{p}_k \]

be a coinductive declaration.

· The red type declaration \( \text{Decl}^r(I) \) for \( I \) is

\[ I^r : \forall \vec{p} : \vec{\rho}. \forall \vec{a} : \vec{\alpha}. \ast, \]
\[ \iota_I : \forall \vec{p} : \vec{\rho}. \forall \vec{a} : \vec{\alpha}. I\vec{p}\vec{a} \rightarrow I^r \vec{p}\vec{a}, \]
\[ \iota^g_I : \forall \vec{p} : \vec{\rho}. \forall \vec{a} : \vec{\alpha}. I^g\vec{p}\vec{a} \rightarrow I^r \vec{p}\vec{a}. \]
First coinduction principle

- Let
  \[ I(\vec{p} : \vec{\rho}) : \forall \vec{a} : \vec{\alpha}.* := \]
  \[ c_1 : \forall \vec{x}_1 : \tau_1. I\vec{p}\vec{u}_1 | \ldots | c_k : \forall \vec{x}_k : \tau_k. I\vec{p}\vec{u}_k \]
  be a coinductive declaration.

- The **red type declaration** \( \text{Decl}^r(I) \) for \( I \) is
  \[
  I^r : \forall \vec{p} : \vec{\rho}. \forall \vec{a} : \vec{\alpha}.*, \\
  \iota_I : \forall \vec{p} : \vec{\rho}. \forall \vec{a} : \vec{\alpha}. I\vec{p}\vec{a} \rightarrow I^r \vec{p}\vec{a}, \\
  \iota^g_I : \forall \vec{p} : \vec{\rho}. \forall \vec{a} : \vec{\alpha}. I^g \vec{p}\vec{a} \rightarrow I^r \vec{p}\vec{a}.
  \]

- The **green type declaration** \( \text{Decl}^g(I) \) for \( I \) is
  \[
  I^g(I^r : \tau_{I^r})(\vec{p} : \vec{\rho}) : \forall \vec{a} : \vec{\alpha}.* := \]
  \[ c^g_1 : \forall \vec{x}_1 : \tau_1[I^r/I]. I^g I^r \vec{p}\vec{u}_1 | \ldots | c^g_k : \forall \vec{x}_k : \tau_k[I^r/I]. I^g I^r \vec{p}\vec{u}_k \]
  where \( \tau_{I^r} = \forall \vec{p} : \vec{\rho}. \forall \vec{a} : \vec{\alpha}.* \) is the arity of the red type \( I^r \).

- For readability, we omit the \( I^r \) parameter to \( I^g \).
First coinduction principle

- Let \( \varphi = \forall \vec{x} : \vec{\tau}.I\vec{u} \) be a first-order type (no quantification over types, propositions, predicates, functions into Type, \ldots).
First coinduction principle

- Let \( \varphi = \forall \vec{x} : \vec{\tau}.I\vec{u} \) be a first-order type (no quantification over types, propositions, predicates, functions into Type, \ldots).
- Let \( \Gamma \) be a first-order context and \( E \) a first-order environment.
First coinduction principle

- Let $\varphi = \forall \vec{x} : \vec{\tau}. I \vec{u}$ be a first-order type (no quantification over types, propositions, predicates, functions into Type, ...).
- Let $\Gamma$ be a first-order context and $E$ a first-order environment.
- Assume $E, \text{Decl}^g(I); \Gamma, \text{Decl}^r(I) \vdash t : \varphi(I^r) \to \varphi(I^g)$.

$\text{Decl}^g(I)$ and $\text{Decl}^r(I)$ denote the declaration of the inductive and coinductive types $I$ respectively.
First coinduction principle

- Let $\varphi = \forall \vec{x} : \vec{\tau}. I \vec{u}$ be a \textit{first-order} type (no quantification over types, propositions, predicates, functions into Type, ...).
- Let $\Gamma$ be a \textit{first-order} context and $E$ a \textit{first-order} environment.
- Assume $E, \text{Decl}^g(I); \Gamma, \text{Decl}^r(I) \vdash t : \varphi(I^r) \rightarrow \varphi(I^g)$.
- Let $t'$ be the normal form of $t$. 
First coinduction principle

- Let $\phi = \forall \vec{x} : \vec{\tau}.I\vec{u}$ be a first-order type (no quantification over types, propositions, predicates, functions into Type, ...).
- Let $\Gamma$ be a first-order context and $E$ a first-order environment.
- Assume $E, \text{Decl}^g(I); \Gamma, \text{Decl}^r(I) \vdash t : \phi(I^r) \rightarrow \phi(I^g)$.
- Let $t'$ be the normal form of $t$.
- Assume $t'$ satisfies the weak case restriction.
First coinduction principle

- Let $\varphi = \forall \vec{x} : \vec{\tau}. I \vec{u}$ be a first-order type (no quantification over types, propositions, predicates, functions into Type, ...).
- Let $\Gamma$ be a first-order context and $E$ a first-order environment.
- Assume $E, \text{Decl}^g(I); \Gamma, \text{Decl}^r(I) \vdash t : \varphi(I^r) \rightarrow \varphi(I^g)$.
- Let $t'$ be the normal form of $t$.
- Assume $t'$ satisfies the weak case restriction.
- Then
  \[ E; \Gamma \vdash \text{cofix}(t'') : \varphi(I) \]
  where

  \[ t'' = t'[I/I^r, \text{id}/\nu_I, \text{id}/\nu^g_I, I/I^g, c_1/c^g_1, \ldots, c_k/c^g_k] \]
  
  and $\text{id} = \lambda \vec{p}. \lambda \vec{a}. \lambda x : I \vec{p} \vec{a}. x$ and $c_1, \ldots, c_k$ are the only constructors of $I$. 
The translation – example

\cdot Let \( I : \ast := c : I \to I \) and \( R : I \to \ast := r : \forall x : I.Rx \to R(cx) \).
The translation – example

- Let $I : * := c : I \rightarrow I$ and $R : I \rightarrow * := r : \forall x : I.Rx \rightarrow R(cx)$.
- Then a proof

$$
\lambda f : (\forall x : I.R^{r}x).\lambda x : I.\text{case}(x, \lambda x.R^{g}x, \lambda x'.r^{g}x'(fx'))
$$

of $(\forall x : I.R^{r}x) \rightarrow \forall x : I.R^{g}x$ gets translated to a syntactically guarded proof

$$
\text{cofix}(\lambda f : (\forall x : I.Rx).\lambda x : I.\text{case}(x, \lambda x.Rx, \lambda x'.rx'(fx'))).
$$

of $\forall x : I.Rx$. 
Correctness of the translation

- Let $\varphi, \Gamma, E$ be first-order.
- Assume $E, \text{Decl}^g(I); \Gamma, \text{Decl}^r(I) \vdash t : \varphi(I^r) \rightarrow \varphi(I^g)$.
- Let $t'$ be the normal form of $t$.
- Assume $t'$ satisfies the weak case restriction.
- Then $E; \Gamma \vdash \text{cofix}(t'') : \varphi(I)$.
Correctness of the translation

- Let $\varphi, \Gamma, E$ be first-order.
- Assume $E, \text{Decl}^g(I); \Gamma, \text{Decl}^r(I) \vdash t : \varphi(I^r) \rightarrow \varphi(I^g)$.
- Let $t'$ be the normal form of $t$.
- Assume $t'$ satisfies the weak case restriction.
- Then $E; \Gamma \vdash \text{cofix}(t'') : \varphi(I)$.

Proof.

- By induction on $t'$.
Correctness of the translation

- Let $\varphi, \Gamma, E$ be first-order.
- Assume $E, \text{Decl}^g(I); \Gamma, \text{Decl}^r(I) \vdash t : \varphi(I^r) \rightarrow \varphi(I^g)$.
- Let $t'$ be the normal form of $t$.
- Assume $t'$ satisfies the weak case restriction.
- Then $E; \Gamma \vdash \text{cofix}(t'') : \varphi(I)$.

Proof.

- By induction on $t'$.
- The weak case restriction allows us to partially recover the subformula property for normal proofs of first-order statements.
Correctness of the translation

- Let $\varphi, \Gamma, E$ be first-order.
- Assume $E, \text{Decl}^g(I); \Gamma, \text{Decl}^r(I) \vdash t : \varphi(I^r) \rightarrow \varphi(I^g)$.
- Let $t'$ be the normal form of $t$.
- Assume $t'$ satisfies the weak case restriction.
- Then $E; \Gamma \vdash \text{cofix}(t''): \varphi(I)$.

Proof.

- By induction on $t'$.
- The weak case restriction allows us to partially recover the subformula property for normal proofs of first-order statements.
- Tedious to carry out this proof in detail, but not mathematically difficult.
How severe are the restrictions?

Short answer: not very.
How severe are the restrictions?

Short answer: not very.

- The **first-order restriction**: the translation often works for statements not satisfying the first-order restriction; then there just is no guarantee that the resulting proof term will be syntactically guarded.
How severe are the restrictions?

Short answer: not very.

- The **first-order restriction**: the translation often works for statements not satisfying the first-order restriction; then there just is no guarantee that the resulting proof term will be syntactically guarded.

  - Let $I : \ast := c : I \to I$ and $R : I \to \ast := r : \forall x : I. Rx \to R(cx)$ be coinductive types.
How severe are the restrictions?

Short answer: not very.

- The **first-order restriction**: the translation often works for statements not satisfying the first-order restriction; then there just is no guarantee that the resulting proof term will be syntactically guarded.
  - Let $I : * := c : I \to I$ and $R : I \to * := r : \forall x : I.Rx \to R(cx)$ be coinductive types.
  - Assume $F : \forall A : *.A \to A$. 
How severe are the restrictions?

Short answer: not very.

- The **first-order restriction**: the translation often works for statements not satisfying the first-order restriction; then there just is no guarantee that the resulting proof term will be syntactically guarded.
  - Let $I : \ast := c : I \to I$ and $R : I \to \ast := r : \forall x : I. Rx \to R(cx)$ be coinductive types.
  - Assume $F : \forall A : \ast. A \to A$.
  - Then

$$\text{cofix}(\lambda f : \forall y. Ry.\lambda y.\text{case}(y, \lambda y. Ry, \lambda x. rx(F(Rx)(fx))))$$

may be obtained using the first coinduction principle.
How severe are the restrictions?

Short answer: not very.

- The **first-order restriction**: the translation often works for statements not satisfying the first-order restriction; then there just is no guarantee that the resulting proof term will be syntactically guarded.
  - Let $I : \star := c : I \to I$ and $R : I \to \star := \forall x : I. Rx \to R(cx)$ be coinductive types.
  - Assume $F : \forall A : \star. A \to A$.
  - Then

\[
\text{cofix}(\lambda f : \forall y. Ry. \lambda y. \text{case}(y, \lambda y. Ry, \lambda x. rx(F(Rx)(fx))))
\]

may be obtained using the first coinduction principle.

- The **weak case restriction**: satisfied by most practically occurring proofs.
How severe are the restrictions?

Short answer: not very.

- **The first-order restriction**: the translation often works for statements not satisfying the first-order restriction; then there just is no guarantee that the resulting proof term will be syntactically guarded.
  - Let $I : * := c : I \to I$ and $R : I \to * := r : \forall x : I. Rx \to R(cx)$ be coinductive types.
  - Assume $F : \forall A : *.A \to A$.
  - Then

$$\text{cofix}(\lambda f : \forall y.Ry.\lambda y.\text{case}(y, \lambda y.Ry, \lambda x.rx(F(Rx)(fx))))$$

may be obtained using the first coinduction principle.

- **The weak case restriction**: satisfied by most practically occurring proofs.
  - Important exception: many proofs using the setoid library for rewriting.
The second coinduction principle

If

\[ \varphi = \forall x_1 : \tau_1 \ldots \forall x_m : \tau_m. \exists y : It_1 \ldots t_p. I_1 s_1^1 \ldots s_{k_1}^1 y \land \ldots \land I_n s_1^n \ldots s_{k_n}^n y \]

where \( y \) does not occur in \( s_{i_j}^j \), then by \( \varphi(I'; I_1', \ldots, I'_n) \) we denote \( \varphi \) 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).
The second coinduction principle

If

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

where \( y \) does not occur in \( s_i^j \), then by \( \varphi(I'; I_1', \ldots, I_n') \) we denote \( \varphi \) 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).

Principle (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.
Coq plugin

```
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.
```
Coq plugin

CoInductive Lex (R : relation nat) :
  Stream nat -> Stream nat -> Prop :=
  | lex_1 : forall x y s1 s2,
    R x y -> Lex R (cons x s1) (cons y s2)
  | lex_2 : forall x s1 s2, Lex R s1 s2 ->
    Lex R (cons x s1) (cons x s2).

CoFixpoint plus s1 s2 := match s1, s2 with
  | cons x1 t1, cons x2 t2 => cons (x1 + x2) (plus t1 t2) end.

Lemma lem_plus : forall x y s1 s2,
  plus (cons x s1) (cons y s2) = cons (x + y) (plus s1 s2).
Proof. peek_eq. Qed.

CoInduction lem_monotone :
  forall (s1 s2 t1 t2 : Stream nat),
  Lex lt s1 t1 -> Lex lt s2 t2 ->
  Lex lt (plus s1 s2) (plus t1 t2).
Proof. destruct 1, 1; do 2 rewrite lem_plus; ccrush. Qed.
Conclusion

- A new coinduction principle for Coq.
Conclusion

- A new coinduction principle for Coq.
  - Ensures guarded use of the coinductive hypothesis.
Conclusion

- A new coinduction principle for Coq.
  - Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.

Coq plugin available: https://github.com/lukaszcz/coinduction.
Conclusion

- A new coinduction principle for Coq.
  - Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.
  - No reformulation of existing coinductive definitions or proofs necessary.

Coq plugin available: [https://github.com/lukaszcz/coinduction](https://github.com/lukaszcz/coinduction).
Conclusion

- A new coinduction principle for Coq.
  - Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.
  - No reformulation of existing coinductive definitions or proofs necessary.
  - Implemented by a direct translation to syntactically guarded Coq proof terms.
Conclusion

- A new coinduction principle for Coq.
  - Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.
  - No reformulation of existing coinductive definitions or proofs necessary.
  - Implemented by a direct translation to syntactically guarded Coq proof terms.
  - Theoretical correctness guarantees when the statement is first-order and the proof satisfies the weak case restriction.

Coq plugin available: https://github.com/lukaszcz/coinduction.
Conclusion

- A new coinduction principle for Coq.
  - Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.
  - No reformulation of existing coinductive definitions or proofs necessary.
  - Implemented by a direct translation to syntactically guarded Coq proof terms.
  - Theoretical correctness guarantees when the statement is first-order and the proof satisfies the weak case restriction.
- Close to informal “pen-and-paper” coinductive reasoning.

Coq plugin available: https://github.com/lukaszcz/coinduction.
Conclusion

- A new coinduction principle for Coq.
  - Ensures guarded use of the coinductive hypothesis.
  - Interacts well with generic automated tactics.
  - No reformulation of existing coinductive definitions or proofs necessary.
  - Implemented by a direct translation to syntactically guarded Coq proof terms.
  - Theoretical correctness guarantees when the statement is first-order and the proof satisfies the weak case restriction.

- Close to informal “pen-and-paper” coinductive reasoning.