# Proof Engineering for Program Logics in Isabelle/HOL

Lecture 3: Introduction to Coinduction in Isabelle

Chelsea Edmonds
University of Western Australia
chelsea.edmonds@uwa.edu.au

ANU Logic Summer School 2025

## Course Overview

**Lectures:**

- Basic reasoning on programs in Isabelle/HOL
- Program Logics: Hoare and Rely-Guarantee
- **A side quest: Intro to Coinduction in Isabelle/HOL**
- Formally defining Rely-guarantee reasoning
- Modular Proofs in Isabelle/HOL

Mix of theory and Isabelle/HOL implementations/proofs.

# Lecture 3 Overview

- Revisiting Induction
- Induction to Coinduction
- Coinductive definitions in Isabelle
- Proofs using coinduction.

# Induction to Coinduction

# Back to Induction

Last week we learnt about:

- Standard inductive data types in Isabelle/HOL

```
datatype 'a list = Nil | Cons 'a "'a list"
```

- Inductive predicates

```
inductive subseq :: "'a list ⇒ 'a list ⇒ bool" where
  ss_empty: "subseq [] xs"
| ss_keep: "subseq xs ys ⟹ subseq (x#xs) (x#ys)"
| ss_drop: "subseq xs ys ⟹ subseq xs (y#ys)"
```

Of course, our semantics and hoare logic definitions are also examples of inductive predicates!

# Breaking Down Induction

So what actually is an inductive definition?

# Breaking Down Induction

Inductive definitions *build up a set incrementally*, starting from a base case.

Think of it as an *iterative* process:

- We start with the empty set
- We keep adding elements according to the rules of the inductive definition.
- Until we reach a *fixed point* - i.e. no new elements can be added.

i.e. an inductive definition is the *smallest set* closed *forward* under its defining rules.

# Breaking Down Induction: Example

Lets take a (slightly simpler) inductive definition for list prefixes.

```
inductive prefix :: "'a list ⇒ 'a list ⇒ bool" where
  pempty: "prefix [] xs"
| pkeep: "prefix xs ys ⟹ prefix (x#xs) (x#ys)"
```

could more traditionally be defined by the following rules using inference rule notation:

$$\frac{}{\mathit{prefix}\ []\ \mathit{xs}}\,(\text{pempty}) \qquad \frac{\mathit{prefix}\ \mathit{xs}\ \mathit{ys}}{\mathit{prefix}\ (x\#\mathit{xs})\ (x\#\mathit{ys})}\,(\text{pkeep})$$

# Breaking Down Induction: Example

$$\frac{}{\textit{prefix}\ []\ \textit{xs}}\text{(pempty)} \qquad \frac{\textit{prefix xs ys}}{\textit{prefix}\ (x\#\textit{xs})\ (x\#\textit{ys})}\text{(pkeep)}$$

In this example, we build up a set of list prefixes by:

- Starting with the empty set
- Adding the empty list
- Incrementally apply the pkeep rule to create more list prefixes.
- Until we reach a fixed point - i.e. no *new* prefixes can be added.

We go from premises (above the line) to conclusions.

# Induction to Coinduction

# Why Coinductive?

Coinductive definitions are traditionally used for defining and reasoning on possibly infinite data structures, e.g.:

- Streams
- Lazy lists
- Infinite trees
- Extended natural numbers

## Coinductive Example

Consider Lazy Lists (i.e. possibly infinite lists).

Say we defined lazy prefix ordering inductively as before:

$$\frac{}{\mathit{lprefix}\ []\ \mathit{xs}}(\text{lpempty}) \qquad \frac{\mathit{lprefix}\ \mathit{xs}\ \mathit{ys}}{\mathit{lprefix}\ (x\#\mathit{xs})\ (x\#\mathit{ys})}(\text{lpkeep})$$

What's the issue with this approach?

# Coinductive Example.

Consider Lazy Lists (i.e. possibly infinite lists).

Say we defined lazy prefixes inductively as before:

$$\frac{}{\mathit{lprefix}\ []\ \mathit{xs}}(\text{ssl\_empty}) \qquad \frac{\mathit{lprefix}\ \mathit{xs}\ \mathit{ys}}{\mathit{lprefix}\ (x\#\mathit{xs})\ (x\#\mathit{ys})}(\text{ssl\_keep})$$

It restricts us to only finite prefixes! No infinite list would be a prefix of any (possibly infinite) list, as an infinite list can't be built up from the base case.

# The Intuition of Coinduction

Coinduction can be thought of as the dual of induction. We *flip* our direction of thinking:

- We start with the set of all possible objects (including infinite ones)
- We *remove* elements that contradict our coinductive rules.

i.e. a coinductive definition is the *largest* set closed *backward* (or consistent) under its defining rules.

# Coinductive Example

Using a coinductive lazy prefix definition:

$$\frac{}{\mathit{lprefix}\ []\ \mathit{xs}}(\text{lpempty}) \qquad \frac{\mathit{lprefix}\ \mathit{xs}\ \mathit{ys}}{\mathit{lprefix}\ (x\#\mathit{xs})\ (x\#\mathit{ys})}(\text{lpkeep})$$

Backward closure goes from the conclusion to the premises.

- We start with the set of all possible lazy lists
- We remove any lists that don't backwards satisfy one of the rules.

All the finite prefixes are included, but also potentially infinite ones! Thinking about our inferences rules - we allow infinite proof derivation trees.

# Induction vs Coinduction

**Induction**

- Smallest set closed forward under the rules
- Build up incrementally moving from premises to conclusions
- Finite derivation trees: i.e. what can be proved using a finite number of rule applications.

**Coinduction**

- Largest set closed backwards under the rules
- Remove inconsistent elements from set of all objects.
- Possibly infinite derivation trees: i.e. what can be proved using an infinite number of rule applications.

# Coinduction in Isabelle

In Isabelle in addition to datatypes we have the codatatypes for defining coinductive types such as lazy lists.

- Standard list datatype definition using datatype

  ```
  datatype 'a list = Nil | Cons 'a "'a list"
  ```

  i.e. all lists that can be constructed in a finite number of steps using Cons from the empty list Nil

- Lazy list datatype definition using codatatype

  ```
  codatatype 'a llist = LNil | LCons (lhd : 'a)  (ltl : "'a llist")
  ```

  i.e. everything that is Nil or that can be *deconstructed* into a head and tail element.

# Coinduction in Isabelle

Similarly, we have a coinductive definition as a dual to the inductive definition.

- Standard list prefix inductive definition:

```
inductive prefix :: "'a list ⇒ 'a list ⇒ bool" where
  pempty: "prefix [] xs"
| pkeep: "prefix xs ys ⟹ prefix (x#xs) (x#ys)"
| ss_drop: "prefix xs ys ⟹ prefix xs (y#ys)"
```

- Lazy list prefix coinductive definition:

```
coinductive lprefix :: "'a llist ⇒ 'a llist ⇒ bool" where
  lpempty: "lprefix [] xs"
| lpkeep: "lprefix xs ys ⟹ lprefix (x#xs) (x#ys)"
```

We also have corecursive, primcorec, etc.

# Isabelle Demo

# Practical Coinduction proofs

# Revisiting Inductive Proofs

An inductive definition gives us several useful proof principles:

- Introduction rules.
- Case Distinction (elimination) rules
- Induction principle.

# Revisiting Inductive Proofs: Introduction Rules

Using our prefix example again, the original inference rules are actually our *introduction* rules:

$$\frac{}{\mathit{prefix}\;[]\;\mathit{xs}}\,(\text{pempty}) \qquad\qquad \frac{\mathit{prefix}\;\mathit{xs}\;\mathit{ys}}{\mathit{prefix}\;(x\#\mathit{xs})\;(x\#\mathit{ys})}\,(\text{pkeep})$$

# Revisiting Inductive Proofs: Case Distinction

The case distinction rule arises from realising that wherever prefix xs xs' holds, it must have been obtainable by one of the previous rules.

$$\frac{\begin{array}{l}\text{prefix } bs\ bs' \\ \forall as.\ bs = [] \wedge bs' = as \longrightarrow P\ bs\ bs' \\ \forall as\ as'\ a.\ bs = a\#as \wedge bs' = a\#as' \wedge \text{prefix } as\ as' \longrightarrow P\ bs\ bs'\end{array}}{P\ bs\ bs'}\text{(Cases)}$$

## Revisiting Inductive Proofs: Case Distinction

The case distinction rule arises from realising that wherever prefix xs xs' holds, it must have been obtainable by one of the previous rules.

$$\frac{\begin{array}{l}\text{prefix } bs\ bs' \\ \forall as.\ bs = [] \wedge bs' = as \longrightarrow P\ bs\ bs' \\ \forall as\ as'\ a.\ bs = a\#as \wedge bs' = a\#as' \wedge \text{prefix } as\ as' \longrightarrow P\ bs\ bs'\end{array}}{P\ bs\ bs'}\text{(Cases)}$$

So if all cases imply a predicate $P$ on $bs$ and $bs'$, then $P\,bs\,bs'$ must hold.

# Revisiting Inductive Proofs: Inductive Principle

The standard inductive principle enables us to show *every element* of an inductively defined set satisfies some condition, if that condition holds under each rule in our inductive definition.

$$\frac{\begin{array}{l}\text{prefix } bs\ bs' \\ \forall as.\ P\ []\ as \\ \forall as\ as'\ a.\ \text{prefix } as\ as' \wedge P\ as\ as' \longrightarrow P\ (a\#as)\ (a\#as')\end{array}}{P\ bs\ bs'}\text{(Induct)}$$

So if $P$ holds under each rule of our prefix inductive definition, and we know $\textit{prefix}\ bs\ bs'$ (i.e. $bs$ is an element of our inductively built set of prefixes of $bs'$), then $P$ must also hold on $bs\ bs'$!

# Coinductive Proofs

Dually, our coinductive definition provides the following rules:

- Introduction rules.
- Case Distinction (elimination) rules
- Coinductive rule.

The introduction and case distinction rules remain the same as the inductive variant.

# Coinductive Proofs: Coinductive Principle

Where as an inductive principle shows that if a condition holds under each rule, every element of the inductive set must satisfy it, the coinductive principle is the reverse - *showing an element is in the coinductive set*.

$$\frac{\begin{array}{l} P\ cs\ cs' \\ \forall bs\ bs'.\ P\ bs\ bs' \longrightarrow (\exists as.\ bs = [] \wedge bs' = as) \vee \\ \qquad\qquad\qquad\qquad (\exists a\ as\ as'.\ bs = a\#as \wedge bs' = a\#as' \wedge P\ as\ as') \end{array}}{\text{prefix}\ cs\ cs'}\ (\text{Coinduct})$$

# Coinduction in Isabelle: Proofs

Like inductive, coinductive automatically generates our proof rules: introduction, case distinction, and coinductive principle (and similarly for (co)datatypes).

However, there is a little more automated support for *applying* an inductive rule in Isabelle. Coinduction sometimes still requires a little more manual support.

# Isabelle Demo

# Coindution and Induction as Fixed Points

# Well-foundedness of our coinductive intuition

This idea of allowing *infinite* proof derivation trees may not feel particularly well-founded.

But it is indeed backed by *fixed point theory*.

Thinking about induction and coinduction via fixpoints makes:

- the duality of the ideas particularly clear.
- It formally clear where each of our proof rules (intro, case distinction, inductive principle, coinductive principle) comes from.

# Induction and Coinduction as Fixed Points

Via the Knaster-Tarski theorem (lattice theory!), we can more formally think of:

- Induction as the least fixpoint
- Coinduction as the greatest fixpoint

See extra materials for more detail if interested.

# Next Time

**Next Lecture:**

- Rely-Guarantee Semantics 3 ways.
- Soundness and Proof Engineering.

**Isabelle exercises/extended work**

- Andrei Popescu's excellent course on Coinduction (with examples in Isabelle): `https://www.andreipopescu.uk/MGS2021/ISA_course.html`
- Section 4 and 5 of the Isabelle (Co)datatypes tutorial
- The Coinductive AFP entry has numerous examples, including much more on lazy lists!
- This is one (of many!) nice example in a blog post comparing some coinductive definitions across Rocq, Isabelle, and Agda: `https://www.joachim-breitner.de/blog/726-Coinduction_in_Coq_and_Isabelle`