Coinduction and topology: an (un)expected connection

Wojciech Kołowski

17 June 2020

Review of approaches to coinduction

- So far we have seen two approaches to coinduction and bisimulation:
- The LTS approach, in which a coinductive process is (re)presented using a particular kind of machine. This machine can be in any of a number of states and can transition between them by performing an appropriate action.
- The categorical approach, in which we are interested in coalgebras of endofunctors. In the final coalgebra $(\nu F, \alpha)$ the "coinductively" defined object is the carrier νF, and α takes the object apart, splitting into its constituent parts (it can also be seen as performing some observations on νF). Corecursion is a consequence of finality.

The LTS approach: a critique 1/2

- I think that the LTS approach to coinduction and bisimulation is quite bad from the explanatory point of view, for a few reasons:
- First, it obscures the very important duality between induction and coinduction, which everybody wants to learn about instantly upon seeing the name "coinduction".
- Interlude: the right notion of equality for LTSes is of course graph isomorphism, as they are nothing more than labeled graphs – static, immobile objects that prescribe actions and transitions, but don't act and don't transition.

The LTS approach: a critique 2/2

- Second, the idea of bisimulation is a bit ad hoc and circular.
- Bisimulation was advertised in the book as the right notion of behavioural equality of LTSes, but it is in fact the right notion of equality for behaviours of LTSes. The behaviour of an LTS is its dynamic aspect – where the actions and transitions take place.
- However, to formally define such a notion of behaviour, we need coinduction in the first place (or else we will miss "infinite" phenomena). Thus, there is some kind of circularity in explaining coinduction using LTSes, even if only conceptual.

The categorical approach: a critique

- The categorical approach is much better, as it makes the duality between induction and coinduction more explicit and also doesn't give the false impression that coinduction is about automata.
- However, it is not without faults:
- By using the machinery of category theory it makes coinduction seem more magical and arcane than it really is. It is unlikely to be enlightening to ordinary programmers and people with category theory disability.
- It makes the operational and computational aspects of corecursion less explicit.
- It does not provide a nice syntax/notation for corecursive definitions (even though it does provide $\nu X.F(X)$ for objects) – and that's very important! "Notation is the tool of thought", they say.

How to explain coinduction to 5 year olds

- I think that the most natural way of explaining coinduction is to refer to an informal version of the duality with induction and then present lots of examples and exercises to build the right intuitions.
- So, let's do just that!

The duality

feature ↓	induction	coinduction
shape	sum (of products)	product (of sums)
basic activity	construction	deconstruction (observation)
derived activity	deconstruction (observation)	construction
easy to define functions with	inductive domain	coinductive codomain
such that every (co)recursive call	shrinks the principal argument	grows the result
thus these functions are	terminating	productive
evaluation	possibly eager	necessarily lazy
tree height	necessarily finite	possibly infinite
tree width	any	any

Some jargon

- The first four rows in the table can be restated in terms of category theory, logic and programmer's common sense.
- In categorical terms it means that inductives have a "mapping-out" universal property, whereas coinductives have a "mapping-in" universal property.
- In logical terms, we can say that inductives have positive polarity and coinductives have negative polarity.
- For programmers this basically means that inductives are data (like that which can be found in databases), whereas coinductives are interactive processes (like operating systems or web servers).

Some intuitions

- This basically means that inductives are determined by ways of constructing their elements and the elimination principle is a derived notion whose purpose is to say "the only things you can build come from the introduction principle(s) (constructors)".
- This basically means that coinductives are determined by ways of observing their elements and the introduction principle is a derived notion whose purpose is to say "the only things you can observe come from the elimination principle(s)".

Some examples

- The type `bool` is an inductive type with two constructors `true` and `false`. Knowing this we can derive an elimination principle, which amounts to an `if-then-else` construction (but dependently typed!).
- Imagine a type of web servers that can only handle requests for pictures of funny cats (this is the elimination principle). Thus know there must be something in the web server that is responsible for handling these requests and the introduction principle consists of all the possible ways of constructing that thing.

A comparatory example

- How does the duality play out in practice? Let's see an example in Coq!
- The code for this example, named `Snippet1.v`, is available from the GitHub repo of this talk: `https://github.com/wkolowski/Seminar-Bisimulation-and-Coinduction`

A comparatory example (informal)

```
data List a = Nil | Cons a (List a)

sum : List Int -> Int
sum Nil = 0
sum (Cons x xs) = x + sum xs
```

```
data List a = Nil

sum : List Int ->
sum Nil = 0
sum (Cons x xs) =
```

●