1 Introduction

These notes are about bisimulation and coinduction. Bisimulation defines *behavioral equality*. Coinduction is a related proof principle. We introduce bisimulation first, but focus mainly on coinduction.

Section 2 motivates our discussion, then Sections 3 and 4 briefly and formally present bisimulation and coinduction. The purpose of these first sections is to associate the definitions one finds in research articles with intuitive explanations. Section 5 gives a series of exercises on streams. We suggest the reader implements solutions in his or her favorite programming language; for reference, we include solutions in Agda, Coq, Haskell, OCaml, and Typed Racket.[1] Finally, we explore connections to inductive reasoning in Section 6 and conclude with a brief review of the literature in Section 7.

2 Motivation

The intution behind bisimulation and coinduction is that we can reason about unknown or "black box" objects by performing experiments on them. This sort of reasoning is extremely natural! It's just the scientific method; coinduction formalizes the scientific method.

SCIENTIFIC METHOD
PURPOSE
State the problem.
RESEARCH
Find out about the topic.
HYPOTHESIS
Predict the outcome to the problem.
EXPERIMENT
Develop a procedure to test the hypothesis.
ANALYSIS
Record the results of the experiment.
CONCLUSION
Compare the hypothesis to the experiment's conclusion.

Figure 1: The Scientific Method

To summarize, Aristotle's Scientific Method says that we can effectively "investigate phenomena" or "acquire knowledge" by systematically observing them.[2] We use this style of reasoning every day, often without realizing it. Suppose the internet stops working, and doesn't come back on immediately or after you slam on the keyboard a few times. The next step is to make observations: you might check the connection settings in your computer or the physical cable attached to your machine. Albiet informal, this is an experiment.

[1] http://ccs.neu.edu/home/types/resources/notes/coinduction-tutorial/src.zip
[2] http://en.wikipedia.org/wiki/Scientific_method

Coinductive reasoning takes this approach to the extreme, asserting that two unknowns are equal if we cannot tell them apart by experiments. At first this may seem unsatisfactory; "never going wrong" is not the same as "going right". But if we think for a moment, we realize that reasoning by experiment is the only technique we have for understanding most of our world.

For example, consider the question of whether animals[3] feel pain. Can we ever prove an answer? Either way, a proof would need to be in terms of the tools we have for measuring pain, and the only such tool is observing that animal's responses to stimuli. This technique is fundamentally unreliable: even if we see the animal twitch, or measure an electrical impulse in its nervous system, we cannot be certain that the animal felt pain in the abstract sense.

Or suppose your iPhone broke and you sent it to a repair shop. As there is no way to open a Apple product, the only way to can check whether the repaired phone is equal to your old phone is by experimenting through the UI. However, reasoning by experiment is not restrictive in this case. It is precisely the equality that you, the phone user, cares about. In the same way, an explorer in the swamps of Florida might not care to distinguish an alligator from a crocodile. They're dangerous enough to discourage an investigation.

A more serious example of reasoning up-to-experiment is evident in O.K. Bouwsma's answer to the Evil Genius problem [3]. The Evil Genius was imagined by Descartes in his *Meditations* on absolute knowledge [4]. Descartes began with the famous "Cogito ergo sum"[4] and proceeded using his senses to reason about the world. But he later noted that his observations might all be invalid if some evil demon had been manipulating his senses. In 1949, the Dutch philosopher O.K. Bouwsma argued that an evil genius intent on confounding our senses could not succeed. If his illusions were not perfect, we could of course sense a flaw and escape. But if he truly was an evil genius and created the perfect illusion, then it would by definition be indistinguishable from reality. Therefore we can live as though there was no illusion at all. Bouwsma's insight was that equality up to experiment is precisely the equality to use for a happy life.

We hope these examples have convinced you that reasoning by experiment / systematic observation / scientific reasoning is:

- Extremely common
- Useful for abstracting unnecessary details
- Quite often the *only* tool at our disposal

The exceptions seem to be finite constructions which, fortunately for us,[5] abound in mathematics and computer science. We know literally everything about a well-founded set, finite binary search tree, or even a Bloom filter. This allows a different style of reasoning: proof by induction. We will consider similarities between induction and coinduction in Section 6. Also see [7].

3 Bisimulation

Bisimulation is a notion of equality. It asserts that two things that act the same are equal.

We use automata as a formal example. Let $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ denote an automata over the alphabet Σ having the set of states Q, the initial state q_0, the set $F \subseteq Q$ of accepting states, and the transition function $\delta : Q \times \Sigma \to Q$ mapping a state to its successor for a given input. Intuitively, $\delta(q, c) = q'$ says that the automata currently in state q will transition upon reading character $c \in \Sigma$ to state q'.

3.1 Definition

A bisimulation on two automata M_1 and M_2 is a relation R on the states of these automata that is preserved by transitions. That is, if the states $q_1 \in Q_1$ and $q_2 \in Q_2$ are related by R, then the following must hold:

- If there is a transition $\delta_1(q_1, c_1) = q_1'$ for any $c \in \Sigma_1$ and $q_1' \in Q_1$, then there exists a character $c_2 \in \Sigma_2$ and state $q_2' \in Q_2$ such that $\delta_2(q_2, c_2) = q_2'$ and $(q_1, q_2) \in R$).

[3]Or cats, or fish, or insects. Pick whatever's interesting.

[4]I am thinking, therefore I exist

[5]i.e., for the author and his intended audience

- If there is a transition $\delta_2(q_2, c_2) = q_2'$ then there is a corresponding transition $\delta_1(q_1, c_1) = q_1'$ and also q_1' and q_2' are related by R.

In other words, the first automata transitions only when the second does, and vice-versa. They move in lock-step. Similar definitions appear, for example, in works on concurrency [12], parametricity [17], and probabilistic programming [9, 5]. We can model these uniformly as *labelled transition systems* [11] consisting of a set of states Q and a function $\delta : Q \times \Sigma \to Q$ defining transitions labelled by elements of an alphabet Σ.

3.2 Usage

One useful bisimulation R is the equivalence class of states based on their accept behavior. In this case, bisimilar automata recognize the same language and are thus equal according to the traditional notion. We can then prune unneeded states from a given machine M by converting it to smaller, bisimilar automata.

In fact, this is the maximal bisimulation on automata and exactly the algorithm suggested by the Myhil-Nerode theorem for regular languages. Briefly, the theorem states that a regular language consists of finitely many words that are pairwise indistinguishable under concatenation on the right. A simple algorithm for constructing a minimal automata is then to consider accepting states distinguishable from rejecting states and then iteratively explore transitions, merging states that transition to indistinguishable states.

The algorithm's correctness follows from the coinduction proof principle.

4 Coinduction

A useful property of bisimulations is their closure under union. If R_1 and R_2 are bisimulations, then $R_1 \cup R_2$ is also a bisimulation. Going further, the union of all bisimulations on a machine M (or any labelled transition system) is a bisimulation. Moreover, it is the *final* bisimulation on M because every other bisimulation is contained in it. This finality leads to the principle of coinduction.

More generally, a coalgebra is a set C, called the carrier of the coalgebra, and a mapping α that gives some structure to elements of C. To be fully precise, a coalgebra (C, α) is defined in terms of some underlying functor F. The type of α is determined by this functor: $\alpha : C \to F(C)$.

$$\begin{tikzcd}
C \arrow[r, "\alpha"] & F(C)
\end{tikzcd}$$

Figure 2: A coalgebra

Intuitively, C is our black box, unknown object. The only thing we know about C is what we can learn through experiments using α. The possible results of any experiment are characterized by F.

Coinduction plays an important role when there exists a final coalgebra $(C_\top, \alpha_\top)$ for F. By final, we mean that there exists a unique coalgebra morphism ρ to $(C_\top, \alpha_\top)$ from any other coalgebra (D, β) that commutes with the maps $\alpha_\top$, β, and F.

$$\begin{tikzcd}
D \arrow[r, dashed, "\rho"] \arrow[d, "\beta"'] & C_\top \arrow[d, "\alpha_\top"] \\
F(D) \arrow[r, "F(\rho)"] & F(C_\top)
\end{tikzcd}$$

Figure 3: A coalgebra homomorphism

Finality implies that the homomorphism ρ both exists and is unique. This leads to two complementary and powerful concepts:

Definition by coinduction follows from the existence of ρ. Given any other coalgebraic structure, there is a homomorphism from it to the final coalgebra.

Proof by coinduction follows from uniqueness. If we can show two maps from a coalgebra to the final coalgebra, then it follows that they are equal.

5 Example: Streams

A stream is an infinite sequence. We claim that streams over a set A, written as $A^{\mathbb{N}}$ and with the structure $\langle \mathsf{head}, \mathsf{tail} \rangle$, are the final coalgebra for the functor $F(X) = A \times F(X)$. To be precise, we claim that the final coalgebra is $(A^{\mathbb{N}}, \langle \mathsf{head}, \mathsf{tail} \rangle)$, where $\mathsf{head} : C \to A$ returns the first element of the stream and $\mathsf{tail} : C \to C$ drops the first element and returns the remaining stream.

$$\mathsf{head}(a) = a(0) \qquad\qquad \mathsf{tail}(a)(n) = a(n+1)$$

We can prove finality by considering an arbitrary coalgebra (D, β), where $\beta : D \to A \times D$, for the functor $F(X) = A \times X$. Because the domain of F is a product, β must have the form $\langle \mathsf{obs}, \mathsf{nxt} \rangle$ for functions $f : D \to A$ and $g : D \to D$. So we have the following situation for an unknown morphism ρ.

$$\begin{tikzcd}
D \arrow[r, dashed, "\rho"] \arrow[d, "{\langle \mathsf{obs}, \mathsf{nxt} \rangle}"'] & A^{\mathbb{N}} \arrow[d, "{\langle \mathsf{head}, \mathsf{tail} \rangle}"] \\
A \times D \arrow[r, "{\langle \mathsf{id}, \rho \rangle}"] & A \times A^{\mathbb{N}}
\end{tikzcd}$$

Figure 4: A coalgebra homomorphism

Now we must define ρ. We choose:

$$\rho(a)(n) = \mathsf{obs}(\mathsf{nxt}^n(a))$$

In other words, the function that takes n "steps forward" before making an observation. Note that this type-checks because $A^{\mathbb{N}}$ is just $\mathbb{N} \to A$.

The first step in the proof is showing that ρ is actually a homomorphism. We show this by observing that $\mathsf{head} \circ \rho = \mathsf{obs}$ and $\mathsf{tail} \circ \rho = \rho \circ \mathsf{nxt}$. Taking an arbitrary $a \in A$, we have:

$$\begin{array}{rcl}
(\mathsf{head} \circ \rho)(a) & = & \mathsf{head}((\lambda\, x\, n \,.\, \mathsf{obs}(\mathsf{nxt}^n(x)))(a)) \\
& = & \mathsf{head}(\lambda\, n \,.\, \mathsf{obs}(\mathsf{nxt}^n(a))) \\
& = & (\lambda\, y \,.\, y(0))\ (\lambda\, n \,.\, \mathsf{obs}(\mathsf{nxt}^n(a))) \\
& = & (\lambda\, n \,.\, \mathsf{obs}(\mathsf{nxt}^n(a)))(0) \\
& = & \mathsf{obs}(\mathsf{nxt}^0(a)) \\
& = & \mathsf{obs}(a)
\end{array}
\qquad
\begin{array}{rcl}
(\mathsf{tail} \circ \rho)(a) & = & \mathsf{tail}((\lambda\, x\, m \,.\, \mathsf{obs}(\mathsf{nxt}^m(x)))(a)) \\
& = & (\lambda\, y\, n \,.\, y(n+1))((\lambda\, m \,.\, \mathsf{obs}(\mathsf{nxt}^m(a)))) \\
& = & \lambda\, n \,.\, (\lambda\, m \,.\, \mathsf{obs}(\mathsf{nxt}^m(a)))(n+1) \\
& = & \lambda\, n \,.\, \mathsf{obs}(\mathsf{nxt}^{n+1}(a)) \\
& = & (\lambda\, z\, n \,.\, \mathsf{obs}(\mathsf{nxt}^n(z)))\ \mathsf{nxt}(a) \\
& = & (\mathsf{tail} \circ \mathsf{nxt})(a)
\end{array}$$

Next is uniqueness. Suppose we have some ρ' such that $\mathsf{head} \circ \rho' = \mathsf{obs}$ and $\mathsf{tail} \circ \rho' = \rho' \circ \mathsf{nxt}$. But we just proved that $\mathsf{head} \circ \rho = \mathsf{obs}$, so by transitivity and definition of composition it must be that $\rho = \rho'$.

5.1 Exercises

We can now leverage the coinductive definition principle to define functions on streams. In OCaml, we can represent streams over the set A as a pair containing an element of A and a delayed computation building the rest of the stream.

```
type 'a stream = Pair of 'a * (unit -> 'a stream)
```

We can further define the universal property for streams as taking an arbitrary coalgebra (represented by the maps obs and nxt) and yielding a stream.

```
let rec univ (obs : 'a -> 'b) (nxt : 'a -> 'a) (seed : 'a) : 'b stream
= Pair (obs seed , fun () -> univ obs nxt (nxt seed))
```

Try definining these stream functions using the universal property instead of general recursion.

```
val nats : int stream
(** Stream of natural numbers 0, 1, 2, ... *)

val evens : 'a stream -> 'a stream
(** Stream of elements in even positions of the argument stream *)

val odds : 'a stream -> 'a stream
(** Stream of elements in odd positions of the argument stream *)

val repeat : 'a -> 'a stream
(** Infinite sequence of one element. [repeat true = true, true, ...] *)

val map : ('a -> 'b) -> 'a stream -> 'b stream
(** Apply a function to each element of a stream. *)

let interleave : 'a stream -> 'a stream -> 'a stream
(** Merge two streams into one. [interleave (evens nats) (odds nats) = nats] *)

val zip : 'a stream -> 'b stream -> ('a * 'b) stream
(** Combine two streams into a stream of products, pointwise *)

val suffixes : 'a stream -> 'a stream stream
(** Stream of all suffixes of a stream *)

val prod : 'a stream -> 'b stream -> ('a * 'b) stream stream
(** Cartesian product of two streams *)

val diag : ('a stream) stream -> 'a stream
(** Take the diagonal entries in a 2D matrix of streams *)

val fib : int stream
(** Fibonacci sequence *)

val look_and_say : int -> int stream
(** [look_and_say 1 = 1, 11, 21, 1211, 111221, 312211, ...] *)
```

Try defining a filter function on streams. What goes wrong?

6 Induction

Coinduction is often compared to the dual notion of proof by induction. Just as coinduction arises from the notion of finality among coalgebra structures, induction comes from initiality in algebras.

An algebra (α, A) for some base functor F consists of a carrier set A and a mapping $\alpha : F(A) \to A$. Whereas the functor F in a coalgebra described the possible results of an experiement—the structure we could impose on our black boxes in A—the functor for an algebra describes how to build new members of A from old ones.

$$\begin{tikzcd} F(C) \arrow[rr, "\alpha"] & & C \end{tikzcd}$$

Figure 5: An algebra

Think of α as a function for building new elements of A, and $F(X)$ as a specification for well-formed inputs to α. The formal term for α is a *constructor*; in contrast, our coalgebras used *destructors*.

The principles of definition and proof by induction apply when F has an initial algebra. To give a concrete example, the algebra $([\mathsf{nil}, \mathsf{cons}], \mathbb{N})$ of finite lists of natural numbers is initial for the functor $F(X) = 1 + X$. Proof by induction says that we can decompose any other algebra on lists into a function that appends elements one at a time. This unique homomorphism is often called a *fold*, in contrast to the *unfold* we used to create all our stream function in Section 5.

7 Further Reading

Rutten and Jacobs have an excellent tutorial on coalgebras and coinduction [7]. We strongly recommend it, along with Jacob's unpublished introductory textbook [6] and Rutten's paper on universal coalgebra [11]. Silva's thesis is also a good source of examples [16].

Sangiorgi's introduction motivates the rest of this section [13]. His books are probably great [14, 15].

7.1 Historical Notes

The term "bisimulation" was coined by Milner and Park in their study of communicating processes in the early 1980s [13]. Coinduction was originally formulated by Aczel in terms of bisimulation in 1988 [1]. However the ideas behind coinduction and bisimulation go back a bit further.

Philosophy

Philosophers were the first to formally use bisimulation arguments.

Ehrenfeucht Fraüsse games (1953), otherwise known as "back-and-forth games", consist of two players and a pair of structures. One player's goal is to show the structures are different. He or she does so by making a proposition / forcing a transition. The other player's goal is to prove the structures are identical, and does so by finding a transition to match the opponents. The argument for equality is thus "the opposer could not show otherwise".

Bisimulation arguments are common in studies of Modal logics. A modal logic takes time into account: the premise p suffices to prove the proposition $\diamond\psi$ if there is some path from p to ψ. The idea is that ψ might eventually become true, but we cannot be certain.

During the study of modal logics in the 1970's, it became apparent that plain homomorphisms were too weak to prove equivalences of modal logics. In 1971 Segerberg added backwards-simulation as a condition on homomorphisms. So no matter which of the two logics could "take a step", the homomorphism guaranteed the other could match it. Later, in 1976 VanBentham pioneered the so-called p-relation (or, zig-zag relation) to relate first order logic to modal logic using a total bisimulation.

Computer Science

Automata and bisimulation are closely connected, though the latter technique was formalized much later. Moore's theory of experiments was motivated by the tests an engineer might perform on a malfunctioning heating system, and Kleene's theory of events was developed to model an animal's interaction with its environment. We find the latter particularly interesting: Kleene defined an event as the set of inputs that would trigger it, reasoning that this is how an animal identifies events.

Bisimulation "per se" was developed by Milner in his work on the Calculus of Communicating Systems [8]. Like the philosophers before him, Milner found that traditional definitions of equality were too strict. In this case, he needed to identify programs with slightly different interleavings of commands.

Math

Mathematicians were the last to work formally with bisimulation and coinduction. Aczel's short 1988 book introduces coinduction in terms of graph theory [1]. Barwise and Moss use systems of equations [2].

We suppose Russell's paradox is to blame for mathematician's reluctance to adopt circularity. The belief that "whatever involves all of a collection can not be one of the collection" is widely held, although Russell's type theory is still less popular than set theory [10]. And despite its inelegance, breaking the world into an inductive hierarchy of universes has been an effective way of solving foundational issues.

References

[1] Peter Aczel and Jon Barwise. *Non-well-founded sets*, volume 14. Center for the Study of Language and Information Stanford, CA, 1988.

[2] Jon Barwise and Lawrence Moss. *Vicious circles: on the mathematics of non-wellfounded phenomena*. Center for the Study of Language and Information, 1996.

[3] Oets Kolk Bouwsma. Descartes' evil genius. *The Philosophical Review*, pages 141–151, 1949.

[4] Rene Descartes. Mediations on first philosophy. 1641. *Haldane, ES; Ross, GRT Trans.: The philosophical works of Rene Descartes. Cambridge University Press, Cambridge*, 1931.

[5] Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. *Information and Computation*, 179, 2002.

[6] Bart Jacobs. Introduction to coalgebra. *Towards mathematics of states and observations*, 2012.

[7] Bart Jacobs and Jan Rutten. A tutorial on (co) algebras and (co) induction. *Bulletin-European Association for Theoretical Computer Science*, 62, 1997.

[8] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

[9] Anna Philippou, Insup Lee, and Oleg Sokolsky. Weak bisimulation for probabilistic systems. In *CONCUR*. 2000.

[10] Bertrand Russell. Mathematical logic as based on the theory of types. *American journal of mathematics*, 30, 1908.

[11] Jan JMM Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249, 2000.

[12] Davide Sangiorgi. A theory of bisimulation for the π-calculus. *Acta informatica*, 33, 1996.

[13] Davide Sangiorgi. On the origins of bisimulation and coinduction. *TOPLAS*, 31, 2009.

[14] Davide Sangiorgi. *Introduction to bisimulation and coinduction*. Cambridge University Press, 2012.

[15] Davide Sangiorgi and Jan Rutten. *Advanced topics in bisimulation and coinduction*. Cambridge University Press, 2012.

[16] Alexandra Silva. Kleene coalgebra. 2010. `http://www.alexandrasilva.org/files/thesis.pdf`.

[17] Eijiro Sumii and Benjamin C Pierce. A bisimulation for type abstraction and recursion. *ACM SIGPLAN Notices*, 40, 2005.