Category Theory and Coalgebra, lecture 6: Final Coalgebras, Bisimulations, Coinduction

Jurriaan Rot

April 15, 2020

In the first three lectures, we've treated several examples of coalgebras—with stream systems in detail—and then moved to the general definition of coalgebras. Afterwards we focused on category theory specifically for a bit; and moved back to coalgebra in the last lecture, through the abstract definition of (final) coalgebras and Lambek's lemma. We now return to the theory of coalgebras, and put together a lot of material that we've seen so far. In particular, we'll start by focussing again final coalgebras, summarising some of the earlier material and making a few new observations. Then we move to the abstract notion of bisimulation between coalgebras, which generalises the proof technique that we've seen for streams to arbitrary coalgebras. We apply the notions of final coalgebra and bisimulation to deterministic automata, viewed as coalgebras, to obtain an algorithm for language equivalence of automata.

Final coalgebras and bisimulations are some of the most important bits of the theory of coalgebras; after this lecture, concluding the first part of the course, we have covered the basics of the theory of coalgebra.

1 Final coalgebras

In the lectures so far, we have seen that, given a functor $F \colon \mathcal{C} \to \mathcal{C}$ on a category $\mathcal{C}$, we obtain notions of:

- *F-coalgebras*, which model state based systems;
- *homomorphisms* between F-coalgebras are behaviour-preserving maps;
- *final F-coalgebra* (if it exists), which models the behaviour of all F-coalgebras, and assigns behaviour to states of F-coalgebras.

Recall the definition of final coalgebras: an F-coalgebra (Z, z) is final if for every F-coalgebra (X, f) there is a unique coalgebra homomorphism from (X, f) to

(Z, z), that is, an arrow $\mathsf{beh}\colon X \to Z$ such that the following diagram commutes:

$$\begin{array}{ccc} X & \overset{\mathsf{beh}}{\dashrightarrow} & Z \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle z} \\ F(X) & \xrightarrow[F(\mathsf{beh})]{} & F(Z) \end{array}$$

We call it beh here, for "behaviour", as we think of it as assigning behaviour to coalgebras. This definition was introduced in the previous lecture; we'll cover several examples now. In the remainder of this note we will take the underlying category $\mathcal{C}$ to be the category Set of sets and functions.

Example 1.1 (Stream systems)**.** For the functor $F(X) = A \times X$, F-coalgebras are stream systems over A: pairs $(X, \langle o, t\rangle)$ where X is a set, and $o\colon X \to A$ and $t\colon X \to X$ functions.

We've covered the final coalgebra in lecture 2: it consists of all streams A^ω. Indeed, for a stream system $(X, \langle o, t\rangle)$ we have the following picture:

$$\begin{array}{ccc} X & \overset{\mathsf{beh}}{\dashrightarrow} & A^\omega \\ {\scriptstyle \langle o,t\rangle}\downarrow & & \downarrow{\scriptstyle \langle i,d\rangle} \\ A \times X & \xrightarrow[\mathsf{id}\times\mathsf{beh}]{} & A \times A^\omega \end{array}$$

where $i(\sigma) = \sigma(0)$ and $d(\sigma) = \sigma'$. As equations:

$$\mathsf{beh}(x)(0) = o(x) \qquad \mathsf{beh}(x)' = \mathsf{beh}(t(x))$$

for all $x \in X$. The map beh assigns to every state $x \in X$ the stream generated by that state. For a proof and a lot more details, see the notes of lecture 2.

Example 1.2 (Deterministic systems with termination)**.** For the functor $F\colon \mathsf{Set} \to \mathsf{Set}$, $F(X) = X + 1$, F-coalgebras are ..., well, maps $f\colon X \to X + 1$ (deterministic systems with termination!), and the final coalgebra consists of 'extended natural numbers' $\mathbb{N} + \{\omega\}$. Given such a deterministic system with termination (X, f), the map $\mathsf{beh}\colon X \to \mathbb{N} + \{\omega\}$ assigns to every state the number of steps after which it terminates.

Another nice example is deterministic automata, where the final coalgebra consists of all languages; see the last section of these lecture notes.

1.1 Behavioural equivalence

In all the above examples, the unique coalgebra homomorphism to the final coalgebra maps each state to its "behaviour" in the final coalgebra. Final coalgebra give us a notion of behavioural equivalence: given states $x, y \in X$ of an F-coalgebra (X, f), we say x and y are *behaviourally equivalent* if $\mathsf{beh}(x) = \mathsf{beh}(y)$, where beh is the unique coalgebra homomorphism from (X, f) to the final F-coalgebra. For instance:

- Two states of a stream system are behaviourally equivalent if they represent the same stream.
- Two states of a deterministic system with termination are behaviourally equivalent if they terminate in the same number of steps.
- Two states of a deterministic automaton are behaviourally equivalent if they accept the same language (treated at the end of this lecture).

1.2 Non-existence of certain final coalgebras

One might be tempted to ask: does every functor $F\colon \mathsf{Set} \to \mathsf{Set}$ have a final coalgebra? The answer is no. We'll give an important example here. It makes use of Lambek's lemma, proved last lecture: if (Z, z) is a final coalgebra, then $z\colon Z \to F(Z)$ is necessarily an isomorphism.

Indeed, take F to be the powerset functor $\mathcal{P}\colon \mathsf{Set} \to \mathsf{Set}$. If we would have a final $\mathcal{P}$-coalgebra, that would mean there is a set Z such that Z and $\mathcal{P}(Z)$ are isomorphic, which means a bijection between them. But this can't be: there is no such set! Therefore, $\mathcal{P}$ does not have a final coalgebra.

An important topic in the theory of coalgebras is the study of when final coalgebras exist, and how to construct them. There have been several answers to this, characterising large classes of functors for which final coalgebras do exist. This goes a bit beyond the scope of the current lecture, but we'll return to it later. For now, we will typically work with final coalgebras on a case-by-case basis; and rest assured final coalgebras exist in many of the examples of interest. For the powerset, one typically works with a restricted version, for instance the finite powerset $\mathcal{P}_\omega$, for which a final coalgebra does exist. Once again, we'll return to it later.

2 Bisimulations and bisimilarity

It's nice that we can define behavioural equivalence, but how do we actually *prove* that two states are equivalent? This is, of course, an important problem if we want to analyse state-based systems: it's one of the most fundamental decision problems.

This question leads us to the main concept of today's lecture: the notion of *bisimulation* between coalgebras. Bisimulations are one of the most important concepts in the theory: they correspond to *proofs of behavioural equivalence*. What is nice, is that bisimulation is defined abstractly in terms of coalgebras; by instantiating to a specific functor (such as the one for stream systems, or for automata) one can then derive a concrete proof technique (to prove, for instance, language equivalence of states of an automaton) and actually derive algorithms from this technique!

We'll start with the general definition of bisimulation, show some examples, and then prove that bisimulations are indeed a sound proof technique for

behavioural equivalence; this is sometimes called 'coinduction proof principle'. We use automata as an extended example, and derive an algorithm for language equivalence.

Although it is possible to formulate the notion of bisimulation for coalgebras for functors on arbitrary categories, we'll focus on functors on Set, since this significantly simplifies matters while still conveying the main ideas (and most of the basic examples are on Set anyway). So let $F\colon \mathsf{Set} \to \mathsf{Set}$ be a functor.

Given F-coalgebras (X, f) and (Y, g), a relation $R \subseteq X \times Y$ is a *bisimulation* (between (X, f) and (Y, g)) if there exists a coalgebra structure $c\colon R \to F(R)$ on R such that the following diagram commutes:

$$\begin{array}{ccccc} X & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & Y \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle c} & & \downarrow{\scriptstyle g} \\ F(X) & \xleftarrow[F(\pi_1)]{} & F(R) & \xrightarrow[F\pi_2]{} & F(Y) \end{array}$$

where π_1, π_2 are the projection maps of R. If $(X, f) = (Y, g)$ then we say R is a bisimulation *on* (X, f).

Of particular interest is the *largest* bisimulation (w.r.t subset inclusion) between (X, f) and (Y, g). It is called *bisimilarity*, and is often denoted by $\sim \subseteq X \times Y$. So, $x \sim y$ iff there exists a bisimulation $R \subseteq X \times Y$ such that $x\ R\ y$. If $x \sim y$ then we say x and y are *bisimilar*.

2.1 Stream systems

First, as an example, let's spell out the details in case of our favorite Set functor: $F(X) = A \times X$, where A is a fixed set. And, to slightly simplify notation, we'll just consider bisimulations on a single F-coalgebra, that is, a stream system $\langle o, f\rangle\colon X \to A \times X$. Instantiating the definition: a relation $R \subseteq X \times X$ is a bisimulation if there exists a stream system $\langle o_R, f_R\rangle\colon R \to A \times R$ such that the following diagram commutes:

$$\begin{array}{ccccc} X & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & X \\ {\scriptstyle \langle o,f\rangle}\downarrow & & \downarrow{\scriptstyle \langle o_R,f_R\rangle} & & \downarrow{\scriptstyle \langle o,f\rangle} \\ A \times X & \xleftarrow[\mathsf{id}\times\pi_1]{} & A \times R & \xrightarrow[\mathsf{id}\times\pi_2]{} & A \times X \end{array}$$

So that means we have, for all $(x, y) \in R$,

- $o(x) = o_R((x, y)) = o(y)$
- $f(x) = \pi_1(f_R(x, y))$
- $f(y) = \pi_2(f_R(x, y))$

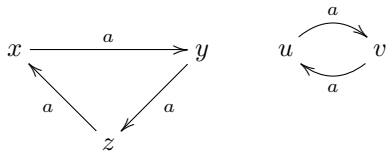
The second and third point together amount to: $f_R(x, y) = (f(x), f(y))$. For this to be correct, we should of course have $(f(x), f(y)) \in R$.

From the above considerations, we can reformulate the notion of bisimulation on our stream system: R is a bisimulation iff for all $(x, y) \in R$:

- $o(x) = o(y)$, and
- $(f(x), f(y)) \in R$.

This concrete characterisation is equivalent to the instance of coalgebraic bisimulation which we started from.

As an example, consider the stream system $\langle o, f \rangle \colon X \to A \times X$, with $X = \{x, y, z, u, v\}$ and the output and transitions given by:



(1)

Let's prove that $x \sim u$. We should come up with a bisimulation R such that $x \; R \; u$. The most economical choice would be $R_0 = \{(x, u)\}$. But does it work? Well, $o(x) = a = o(u)$, so that's fine, but $\ldots (f(x), f(u)) = (y, v) \notin R_0$. So R_0 is not a bisimulation. Ok, but we don't give up: lets take $R_1 = \{(x, u), (y, v)\}$. Then (x, u) is fine, but for (y, v) we have $(f(y), f(v)) \notin R_1$ $\ldots$ and this game continues for a while, until we arrive at

$$R = \{(x, u), (y, v), (z, u), (x, v), (y, u), (z, v)\}$$

and we finally have our desired bisimulation! By the way, which pairs are actually in the *largest* bisimulation $\sim$ in this example?

We conclude our little streams excursion by recovering an old friend from the second lecture: bisimulations between streams. Take the final coalgebra $\langle i, d \rangle \colon A^\omega \to A \times A^\omega$. A relation $R \subseteq A^\omega \times A^\omega$ on it is a bisimulation if for all $(\sigma, \tau) \in R$,

- $i(\sigma) = i(\tau)$, that is, $\sigma(0) = \tau(0)$, and
- $(d(\sigma), d(\tau)) \in R$, that is, $(\sigma', \tau') \in R$.

This is exactly the notion we saw in lecture 2, where we showed for instance that the relation

$$R = \{(even(tail(\sigma)), odd(\sigma)) \mid \sigma \in A^\omega\} \tag{2}$$

is a bisimulation. The point, which we showed in lecture 2, is that, whenever $R \subseteq A^\omega \times A^\omega$ is a bisimulation, then $(\sigma, \tau) \in R$ implies $\sigma = \tau$. Or, phrased differently, the greatest bisimulation $\sim$ is simply the equality relation $\{(\sigma, \sigma) \mid \sigma \in A^\omega\}$. We call this a *coinduction proof principle*. In a moment, we'll see that such a principle holds not only for streams but much more generally, using the final coalgebra.

2.2 Deterministic systems with termination

Of course, if I'm pretending to give a really general definition, we should have at least one more example. So let's consider coalgebras for the functor F given by $F(X) = X + 1$. According to the definition, a relation $R \subseteq X \times X$ on the states of a coalgebra $f\colon X \to X + 1$ is a bisimulation if for all $(x, y) \in R$ (let's assume for simplicity that X is disjoint from $1 = \{*\}$):

- either $f(x) = * = f(y)$, or $(f(x), f(y)) \in R$ (which implicitly means $f(x), f(y) \in X$).

This is equivalent to the coalgebraic definition of bisimulation for this functor; if you're not convinced immediately, working this out is a good exercise. We'll see some further instances in the weekly exercises.

3 Coinduction

So we defined when two states of a coalgebra are bisimilar, denoted $x \sim y$, which means $(x, y) \in R$ for some bisimulation R. But what's the point? Well, first of all, bisimilarity $\sim$ is a really important notion of equivalence in itself. For instance, in concurrency theory, it captures behavioural equivalence of processes; we'll talk about this later in the course. However, as it turns out, within the theory of coalgebras, it is a *proof technique for behavioural equivalence*. This is formally stated by the following result.

Theorem 3.1 (Coinduction proof principle)**.** *Let $F\colon \mathsf{Set} \to \mathsf{Set}$ be a functor which has a final coalgebra (Z, z), and let (X, f) and (Y, g) be F-coalgebras. If $R \subseteq X \times Y$ is a bisimulation, then for all $x \in X$ and $y \in Y$:*

$$(x, y) \in R \quad \Rightarrow \quad \mathsf{beh}_f(x) = \mathsf{beh}_g(y)\,,$$

where $\mathsf{beh}_f\colon X \to Z$ and $\mathsf{beh}_g\colon Y \to Z$ are the unique coalgebra homomorphisms.

Proof. Suppose $(x, y) \in R$, and R is a bisimulation, witnessed by some coalgebra structure $c\colon R \to F(R)$. Consider the following diagram:

$$\begin{array}{ccccccccc}
Z & \xleftarrow{\mathsf{beh}_f} & X & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & Y & \xrightarrow{\mathsf{beh}_g} & Z \\
\downarrow{\scriptstyle z} & & \downarrow{\scriptstyle f} & & \downarrow{\scriptstyle c} & & \downarrow{\scriptstyle g} & & \downarrow{\scriptstyle z} \\
F(Z) & \xleftarrow[F(\mathsf{beh}_f)]{} & F(X) & \xleftarrow[F(\pi_1)]{} & F(R) & \xrightarrow[F\pi_2]{} & F(Y) & \xrightarrow[F(\mathsf{beh}_g)]{} & F(Z)
\end{array}$$

The above diagram commutes: the inner squares since R is a bisimulation (with coalgebra structure c) and the outer two squares since beh_f and beh_g are, by definition, coalgebra homomorphisms of this type.

As a consequence, both $\mathsf{beh}_f \circ \pi_1$ and $\mathsf{beh}_g \circ \pi_2$ are coalgebra homomorphisms from (R, c) to the final coalgebra (Z, z), hence

$$\mathsf{beh}_g \circ \pi_2 = \mathsf{beh}_f \circ \pi_1\,.$$

Thus, for any $(x, y) \in R$, we have $\mathsf{beh}_f(x) = \mathsf{beh}_f(\pi_1(x, y)) = \mathsf{beh}_g(\pi_2(x, y)) = \mathsf{beh}_g(y)$ as needed. □

As a consequence, to show that two states x, y are behaviourally equivalent, it suffices to show that they are related by a bisimulation. For instance, if R is a bisimulation between stream systems, then any $(x, y) \in R$ means that x and y represent the same stream. So in particular, we obtain the rather spectacular fact that x and u in (1) represent the same stream! Ok, that's maybe not the most exciting example; but of course, the technique works for more complicated stream systems as well. One class of examples is the bisimulation proofs on streams which we did in lecture 2, see for instance (2) above. By Theorem 3.1, we recover the correctness of this technique: if $(x, y) \in Z$ are elements of a final coalgebra, then $x = y$ (notice that $\mathsf{beh} = \mathsf{id}$ in that case).

We formally state the above observation in a little theorem, which also includes a converse. In fact, the coinduction proof principle is sometimes stated in the following form: bisimilarity on the final coalgebra coincides with equality.

Theorem 3.2. *Let $F \colon \mathsf{Set} \to \mathsf{Set}$ be a functor which has a final coalgebra (Z, z). Then for all $(x, y) \in Z$: $x \sim y$ iff $x = y$.*

Proof. From left to right, this follows from Theorem 3.1. From right to left, one shows that the relation $\{(x, x) \mid x \in Z\}$ is a bisimulation. In fact, that holds on any coalgebra: a nice exercise. □

The above theorem generalises what we showed in lecture 2: that bisimilarity of streams coincides with equality. Of course, the above theorem is much more general, applying to any kind of coalgebra; for instance, it also allows us to capture equality of languages through bisimilarity on the final deterministic automaton. We'll use this in another lecture; and in a moment, we'll concider bisimulations on (arbitrary) determinstic automata, which, by Theorem 3.1 suffice to prove language equivalence of states.

We finish our discussion of coinduction with a little observation on homomorphisms and bisimilarity. Notice that we obtain (one side of) Theorem 3.2 as a special case Theorem 3.1. As it turns out, one can also go the other way, and obtain Theorem 3.1 from Theorem 3.2. That argument uses that, if h, k are homomorphisms, then $x \sim y$ implies $h(x) \sim k(y)$ — we won't go into details at this point.

However, one may also wonder whether there is a converse: is it the case that $h(x) \sim k(y)$ implies $x \sim y$? And specifically, do we have a converse for Theorem 3.1, that is, does $\mathsf{beh}(x) = \mathsf{beh}(y)$ imply $x \sim y$? This is a reasonable question, as it is a kind of completeness: if it holds, this means that whenever two states are behaviourally equivalent, we can prove this using bisimulations. However, as it turns out, the answer is no: there are some functors for which this is not the case. Fortunately, it holds under mild conditions, and in particular for most functors that we are interested in: for instance, all (Kripke) polynomial functors, that is, functors built using products, coproducts, exponents, constant functors, the identity functor and the (finite) powerset functor. The condition

typically used is that the functor preserves weak pullbacks, but this is a bit beyond the scope of the course right now; if you're interested, have a look, for instance, at [4].

4 Deterministic automata: final coalgebra and bisimulations

In this section we'll treat an important example of the theory of coalgebras: deterministic automata. We'll see how language semantics arises through the final coalgebra, and bisimulation gives a proof method for language equivalence of deterministic automata.

First, we fix some basic definitions and notation. We'll make use of the two-element set $2 = \{0, 1\}$. For an alphabet A, we denote by A^* the set of words, and ε the empty word. A language is a subset of A^*; equivalently, a function $L \colon A^* \to 2$ (a word is in the language if $L(w) = 1$). So the set of all languages is given by 2^{A^*}.

Throughout this section, we fix the functor $F \colon \mathsf{Set} \to \mathsf{Set}$, $F(X) = 2 \times X^A$. A coalgebra for this functor is a set X together with a map $\langle o, \delta \rangle \colon X \to 2 \times X^A$. We say $x \in X$ is accepting if $o(x) = 1$, and we write $x \xrightarrow{a} y$ if $y = \delta(x)(a)$, meaning that x makes an a-transition to y. Thus an F-coalgebra is a *deterministic automaton* (where X is not necessarily finite, and there is no initial state).

4.1 Final coalgebra

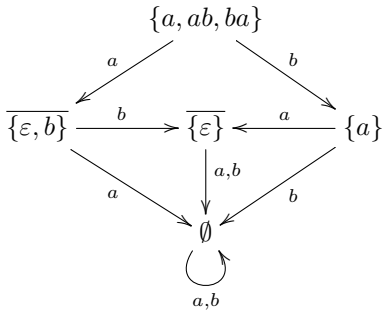
Consider the following F-coalgebra: $(2^{A^*}, \langle e, d \rangle)$ consisting of the set 2^{A^*} of all languages. So: states of this automaton are languages over A^*. The output function is given by $e(L) = L(\varepsilon)$, i.e., $e(L) = 1$ iff L contains the empty word. The transition function, for a language L and letter $a \in A$, is given by $d(L)(a) = L_a$, where L_a is *language derivative*:

$$L_a(w) = L(aw).$$

Thus, a word w is in L_a precisely if the word aw is in L. In other words, to compute L_a, take all words of L that start with an a and chop off that a.

For a basic example, consider the language $\{a, ab, ba\}$ over the alphabet $A = \{a, b\}$. The part of the automaton of languages reachable from the state

$\{a, ab, ba\}$ looks as follows:



We claim that $(2^{A^*}, \langle e, d\rangle)$ is a final F-coalgebra. So, let $(X, \langle o, \delta\rangle)$ be a deterministic automaton. We're after a map beh making the following diagram commute:

$$\begin{array}{ccc} X & \xrightarrow{\mathsf{beh}} & 2^{A^*} \\ {\scriptstyle \langle o,\delta\rangle}\downarrow & & \downarrow{\scriptstyle \langle e,d\rangle} \\ 2 \times X^A & \xrightarrow[\mathsf{id}\times \mathsf{beh}^A]{} & 2 \times (2^{A^*})^A \end{array}$$

What does it mean for such a map beh to make this diagram commute? Well, if we spell this out, we have for all $x \in X$:

$$\mathsf{beh}(x)(\varepsilon) = e(\mathsf{beh}(x)) = o(x)\,,$$

and, for all $a \in A$ and $w \in A^*$:

$$\mathsf{beh}(x)(aw) = \mathsf{beh}(x)_a(w) = d(\mathsf{beh}(x))(a) = \mathsf{beh}^A(\delta(x))(a) = \mathsf{beh}(\delta(x)(a))$$

Exercise: try to derive this from the diagram, definitions of e, d and language derivative yourself!

Thus, putting these together, a map beh makes the diagram above commute precisely if the following equations are satisfied:

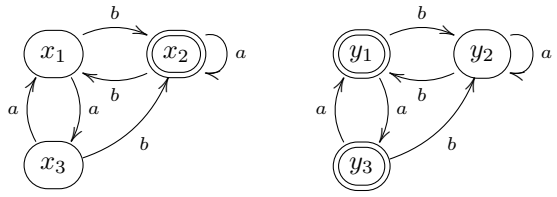
- $\mathsf{beh}(x)(\varepsilon) = o(x)$, and
- $\mathsf{beh}(x)(aw) = \mathsf{beh}(\delta(x)(a))(w)$

for all $x \in X$, $a \in A$, $w \in A^*$. But it follows by induction on $w \in A^*$ that such a map beh exists, and is unique! Indeed, the first equation gives the base case, and the second equation the induction step.

What is beh? It's the language semantics of deterministic automata, mapping every state to the language it accepts! The empty word is accepted precisely from the state x if x is an accepting state $(o(x) = 1)$ and a word of the form aw is accepted from x if $x \xrightarrow{a} y$ for some state y such that y accepts w.

4.2 Bisimulations

The notion of bisimulation will provide us with the means for checking *language equivalence*: whether two states $x, y \in X$ of a deterministic automaton accept the same language, i.e., whether $\mathsf{beh}(x) = \mathsf{beh}(y)$. Consider, for instance, the following automaton (for simplicity of notation, we view the entire thing as a single automaton):



(3)

In this example, x_1, x_3 and y_2 are all behaviourally equivalent; so are x_2, y_1 and y_3. Of course, this is a simple example, just for the sake of argument; for bigger automata, language equivalence becomes much less obvious.

First of all, we'll spell out the notion of bisimulation for deterministic automata. As always, we start by instantiating the definition: a relation $R \subseteq X \times X$ on the states of an automaton $(X, \langle o, \delta \rangle)$ is a bisimulation if there exists a coalgebra structure $\langle o_R, \delta_R \rangle \colon X \to 2 \times X^A$ such that the following diagram commutes:

$$\begin{array}{ccccc} X & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & X \\ {\scriptstyle \langle o,\delta \rangle}\downarrow & & \downarrow{\scriptstyle \langle o_R,\delta_R \rangle} & & \downarrow{\scriptstyle \langle o,\delta \rangle} \\ 2 \times X^A & \xleftarrow[\mathsf{id}\times\pi_1]{} & 2 \times R^A & \xrightarrow[\mathsf{id}\times\pi_2]{} & 2 \times X^A \end{array}$$

So, equivalently, we must have, for all $(x, y) \in R$:

- $o(x) = o_R(x, y) = o(y)$,
- $\forall a. \delta(x)(a) = \pi_1(\delta_R(x, y)(a))$ and
- $\forall a. \delta(y)(a) = \pi_2(\delta_R(x, y)(a))$.

A coalgebra structure $\langle o_R, \delta_R \rangle$ satisfying the above conditions exists if and only if the following holds, for all $(x, y) \in R$:

- $o(x) = o(y)$, and
- $\forall a \in A.\ \delta(x)(a)\ R\ \delta(y)(a)$.

We have obtained a concrete notion of bisimulation for deterministic automata: a relation $R \subseteq X \times X$ satisfying the above two requirements (which don't anymore mention a coalgebra structure!): if two states x, y are related, then they should either both be accepting or both be non-accepting; and for every alphabet letter, the states reached after taking a transition (labelled by that letter) from both states should be again related.

It follows from Theorem 3.1, that if $(x, y) \in R$ for some bisimulation R, then x and y are language equivalent: $\mathsf{beh}(x) = \mathsf{beh}(y)$. Thus, to prove language equivalence, it suffices to construct a suitable bisimulation.

To start with, let's show that x_1, y_2 in (3) are equivalent, using bisimulations. The first attempt could be

$$R_0 = \{(x_1, y_2)\}\,.$$

Both x_1, y_2 are non-accepting. However, $\delta(x_1)(a) = x_3$, whereas $\delta(y_2)(a) = y_2$, so this is not a bisimulation, since $(x_3, y_2) \notin R_0$; also, $(\delta(x_1)(b), \delta(y_2)(b) = (x_2, y_1) \notin R_0$. So we add both pairs, and let

$$R_1 = \{(x_1, y_2), (x_3, y_2), (x_2, y_1)\}$$

This is still not a bisimulation . . . however, after one more step, we reach

$$R_2 = \{(x_1, y_2), (x_3, y_2), (x_2, y_1), (x_2, y_3)\}$$

which is actually a bisimulation. Hence, $\mathsf{beh}(x_1) = \mathsf{beh}(y_2)$, that is, x_1 and y_2 are language equivalent.

Note that we could have (somehow) guessed the entire relation R_2 at once; it's not at all necessary to construct in the above, incremental way. However, what is nice about this, is that it looks pretty algorithmic: one just adds successor states and keeps checking whether both are accepting/non-accepting, until either the relation is a bisimulation, or one has to add some pair (x, y) which violates the first condition, that is, where one is accepting and the other is not. In fact, in the latter case, we have found a counterexample: a reason that the two states are *not* language equivalent.

Not only does this look algorithmic, we can just turn it into an algorithm! To this end, we'll focus now on *finite* automata: we assume that the set of states X is finite. We call the algorithm Naive, as we'll see in a moment how to improve it. This, as well as the adapated algorithm in the next section, is more or less literally copied from [2] and from notes written by Filippo Bonchi and myself (see [1]). Given a deterministic automaton $(X, \langle o, \delta\rangle)$, and states $x_0, y_0 \in X$, the algorithm is as follows:

Naive(x_0, y_0)

```
(1) R is empty; todo is empty;
(2) insert (x0, y0) in todo;
(3) while todo is not empty do
 (3.1)  extract (x, y) from todo;
 (3.2)  if (x, y) ∈ R then continue;
 (3.3)  if o(x) ≠ o(y) then return false;
 (3.4)  for all a ∈ A,
          insert (δ(x)(a), δ(y)(a)) in todo;
 (3.5)  insert (x, y) in R;
(4) return true;
```

First of all, observe that the algorithm is correct: $\texttt{Naive}(x_0, y_0)$ returns true iff $x_0 \sim y_0$ (iff $\mathsf{beh}(x_0) = \mathsf{beh}(y_0)$). To see this, notice that the following is a loop invariant: for all $(x, y) \in R$, we have

- $o(x) = o(y)$, and
- $\forall a \in A.\ (\delta(x)(a), \delta(y)(a)) \in R \cup \mathit{todo}$.

Since in the end todo is empty, it follows that, if the algorithm terminates returning true, the relation R is a bisimulation. Conversely, suppose $\mathsf{beh}(x_0) \neq \mathsf{beh}(y_0)$. Then eventually a pair (x, y) will be added such that $o(x) \neq o(y)$, hence the algorithm returns false, in (3.3).

If the state space X is finite, then the algorithm terminates. To see this, notice first that every iteration of the loop, an element is taken from todo. Further, new elements are added to todo only if some new element is also added to R. Hence, the number of times new elements are added to todo is bounded by the number of pairs of elements of X.

Why is this algorithm called `Naive`, by the way? Our algorithm was derived more or less directly from the instance of coalgebraic bisimulation for deterministic automata. One can think of similar algorithms for other types of coalgebras, just based on the associated notion of bisimulation. This is all nice, but . . . it turns out that, in many cases, one can do better: it is possible to prove behavioural equivalence in an 'easier' way, by slightly relaxing the requirements of a bisimulation. In case of deterministic automata, this leads to a classical algorithm for language equivalence, as was observed in [2]. But we've had enough material in this lecture, and postpone this to a later lecture.

5 Final remarks

We've covered a lot in this lecture: examples of final coalgebras, and the coalgebraic notion of bisimulation. We've mainly seen our favorite instances, streams and automata; however, bisimulation is defined rather generally, in terms of a (Set) functor. For instance, later in the course, we'll consider the case of labelled transition systems, recovering the classical case of bisimulations introduced first, historically (and which is still the most famous instance of bisimulation).

A little remark on terminology: we've been using the word *coinduction* sometimes to *define* maps into a final coalgebra ("coinductive" definition of an operation such as even or odd on streams), and, mainly today, to talk about a *proof* technique. The common theme here is that we refer to coinduction as the "use of final coalgebras"; as, for instance, in [3]. In computer science, there's also a more classical use of the word 'coinduction', in the context of partial orders; we'll see this later in the course.

References

[1] Filippo Bonchi, Marcello Bonsangue, and Jurriaan Rot. Lecture notes: coalgebraic methods for automata, 2018. `http://esslli2018.folli.info/wp-content/uploads/coma.pdf`.

[2] Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2013, proceedings*, pages 457–468. ACM, 2013.

[3] Bart Jacobs and Jan Rutten. An introduction to (co)algebras and (co)induction. In *Advanced Topics in Bisimulation and Coinduction*, pages 38–99. Cambridge University Press, 2012.

[4] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.