Coalgebra, lecture 13
Final sequence

Jurriaan Rot

December 10, 2018

Before, we've seen that every monotone function on a complete lattice has a greatest fixed point. In this lecture, we're going to look at the final sequence, which provides a more concrete way of actually computing greatest fixed points, and underlies algorithms for computing the greatest bisimulation of automata and transition systems, for instance. In the second half of the lecture we'll see that all this can be generalised to categories and coalgebras, giving a recipe for computing final coalgebras (under some conditions on the functor). This second part is not described in these notes; see Section 4.6 of the book by Jacobs.

1 Greatest bisimulation of an automaton

As (almost) always, we'll first look at deterministic automata, and then describe the more general picture in the next section. So let $\langle o, \delta \rangle \colon X \to 2 \times X^A$ be a deterministic automaton, and consider our favorite monotone function $b \colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ defined by

$$b(R) = \{(x, y) \mid o(x) = o(y) \text{ and for all } a \in A \colon (\delta(x)(a), \delta(y)(a)) \in R\}\,.$$

A bisimulation is a post-fixed point of b (an R with $R \subseteq b(R)$) and the greatest fixed point $\mathsf{gfp}(b)$ is the greatest bisimulation. We also know that this coincides with language equivalence.

Before, we've often used this as follows: to show that states $x, y \in X$ are bisimilar (language equivalent), we construct a bisimulation, by starting with $\{(x, y)\}$ and adding pairs until the relation is a bisimulation (or a counterexample is found). Now, rather than computing bisimulations from below (in the lattice of relations), we'll compute the *greatest* bisimulation from above, that is, by starting with a lot of pairs and removing non-bisimilar pairs until we end up with $\mathsf{gfp}(b)$. This addresses a slightly different problem: it computes the greatest bisimulation (rather than showing two specific states to be bisimilar). The greatest bisimulation is a very useful thing to have: by identifying states that are bisimilar, we obtain a *minimal* automaton.

The approach is to start with the relation $X \times X$, relating everything, and then iteratively apply b. First, we compute $b(X \times X)$:

$$b(X \times X) = \{(x, y) \mid o(x) = o(y)\}\,.$$

So that relates everything with the same output. We apply b to the result:

$$\begin{aligned} b(b(X \times X)) &= \{(x, y) \mid o(x) = o(y) \text{ and for all } a \in A\colon (\delta(x)(a), \delta(y)(a)) \in b(X \times X)\} \\ &= \{(x, y) \mid o(x) = o(y) \text{ and for all } a \in A\colon o(\delta(x)(a)) = o(\delta(y)(a))\} \end{aligned}$$

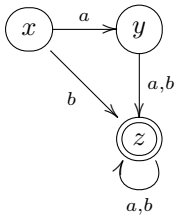
That relates all states which have the same output and the same output after one transition. More generally, let b^i be the i-fold composition of b (formally $b^0 = \mathsf{id}$ and $b^{i+1} = b \circ b^i$). Then $b^i(X \times X)$ relates states (x, y) if, whenever we read a word w of length below i from x and y, we end up with states with the same output. What does this mean in terms of languages?

The above construction gives a decreasing sequence:

$$X \times X \supseteq b(X \times X) \supseteq b(b(X \times X)) \supseteq \ldots \supseteq b^i(X \times X) \supseteq \ldots$$

If X is finite, then this sequence stabilises (stops decreasing) after a finite number of steps (why?). More precisely, then there exists a natural number i such that $b^i(X \times X) = b(b^i(X \times X))$. In that case, we also have $b^i(X \times X) = b^j(X \times X)$ for *all* $j \geq i$. Notice that, when the sequence stabilises at i, $b^i(X \times X)$ is a fixed point; the important observation, which we'll prove in a more general setting in the next section, is that it is the *greatest fixed point of* b. So the above recipe gives us a way of computing the greatest bisimulation of an automaton: start with $X \times X$, and keep applying b until that has no more effect, that is, we reached a fixed point.

As an example, consider the following automaton.



Here $X = \{x, y, z\}$ and we compute the first few steps of the decreasing sequence:

$$\begin{aligned} X \times X &= \ldots \\ b(X \times X) &= \{(x, x), (y, y), (z, z), (x, y), (y, x)\} \\ b(b(X \times X)) &= \{(x, x), (y, y), (z, z)\} \\ b(b(b(X \times X))) &= \{(x, x), (y, y), (z, z)\} \end{aligned}$$

Since $b^2(X \times X) = b^3(X \times X)$, the sequence stabilises after two steps, and we may stop here. We conclude with the relation $\{(x, x), (y, y), (z, z)\}$, which is

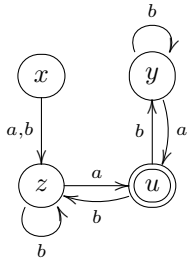
hence the greatest bisimulation. Note that in the second step, (x, y) (and (y, x)) are not related since $x \xrightarrow{a} y$ and $y \xrightarrow{a} z$ but $(y, z) \notin b(X \times X)$.

We have an *equivalence* relation in each step in the above example. In fact, it is not so difficult to prove that, in general, if R is an equivalence relation then $b(R)$ as well, for our choice of b. This allows us to represent the steps in the computation of the decreasing sequence as *partitions* rather than equivalence relations, which is much more useful from a computational (and notational) perspective. For instance, in the above example, the partitions become:

$$\begin{aligned} &\{\{x, y, z\}\} && (X \times X) \\ &\{\{x, y\}, \{z\}\} && (b(X \times X)) \\ &\{\{x\}, \{y\}, \{z\}\} && (b(b(X \times X))) \end{aligned}$$

The above approach for computing the greatest bisimulation (and, hence, language equivalence), with the relations presented as partitions, is called *partition refinement*. We start with the partition containing everything, and refine it at each step according to b, until it stabilises. This means we states x, y end up in the same equivalence class at step $i + 1$ if x and y have the same output and for each letter a, given $x \xrightarrow{a} x'$ and $y \xrightarrow{a} y'$, x' and y' are in the same equivalence class at step i.

As another example, consider the automaton below.



(1)

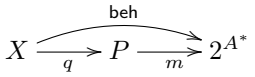
The steps of partition refinement are:

$$\begin{aligned} &\{\{x, y, z, u\}\} \\ &\{\{x, y, z\}, \{u\}\} \\ &\{\{x\}, \{y, z\}, \{u\}\} \\ &\{\{x\}, \{y, z\}, \{u\}\} \end{aligned}$$

and we stop. So y, z are language equivalent.

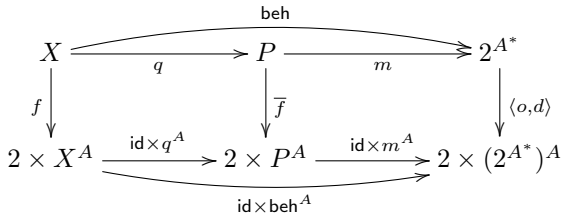
A sensible way of implementing all this is to start with the partition that distinguishes accepting and non-accepting states (corresponding to the relation $b(X \times X)$), and then computing the next step based on inverse images of each equivalence class in the current step, along the transitions. We won't go into details here, but if you're interested: the basic version doing this is called Moore's algorithm. There's also a more advanced version, called Hopcroft's algorithm, which is more involved, but has a better complexity.

Minimisation. With the procedure above, we can compute language equivalence of a given automaton. Once this is done, one nice result is a minimal automaton; we briefly and informally sketch how we get this. To phrase this coalgebraically, let (X, f) be an automaton, and consider the unique coalgebra morphism $\mathsf{beh}\colon X \to 2^{A^*}$ to the final coalgebra. Language equivalence means taking the image of this map: let P be the partition of X which identifies language equivalent states (it can be computed by the algorithm described above), with the quotient map $q\colon X \to P$. Then beh factors as

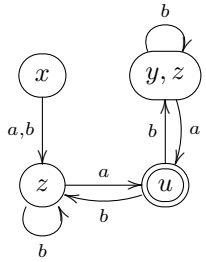


where m is the injective map sending an equivalence class of states to the language they (all) represent.

Now, we can turn P into a coalgebra structure $(P, \overline{f})$, by choosing an arbitrary representative of each equivalence class (yes, this is well-defined). This turns q, m into coalgebra morphisms, and we get a commutative diagram:



The automaton $(P, \overline{f})$ is the minimisation of (X, f). For instance, the DFA described above has the following minimisation:



2 Kleene fixed point theorem

We generalise the computation of the greatest bisimulation of automata from the previous section, to the computation of the greatest fixed point of a monotone function $b\colon P \to P$ on a complete lattice P. In fact, this will require some additional assumptions on b.

The idea is to look at the *final sequence*:

$$\top \geq b(\top) \geq b(b(\top)) \geq \dots$$

where $\top$ is the top element of P. And then take the meet of all those:

$$\bigwedge_{i \in \mathbb{N}} b^i(\top)$$

(formally, we mean $\bigwedge\{b^i(\top) \mid i \in \mathbb{N}\}$). Is this the greatest fixed point? It turns out that this doesn't quite work in general, and we need some additional assumptions on b. We'll leave it as an (optional) exercise to find an example of a monotone b such that $\bigwedge_{i \in \mathbb{N}} b^i(\top) \neq \mathsf{gfp}(b)$. By the way, one eventually reaches the greatest fixed point by applying b some more: by extending it to an ordinal-indexed sequence. We won't treat that here.

A *decreasing sequence* is a sequence $\alpha_0, \alpha_1, \alpha_2, \ldots$ of elements of P such that $\alpha_i \geq \alpha_{i+1}$ for all i. A function $b \colon P \to P$ on a complete lattice P is called *cocontinuous* if for every decreasing sequence $(\alpha_i)_{i \in \mathbb{N}}$:

$$b(\bigwedge_{i \in \mathbb{N}} \alpha_i) = \bigwedge_{i \in \mathbb{N}} b(\alpha_i)\,.$$

Every cocontinuous function is also monotone (but the converse is not true); this is an exercise.

Theorem 2.1 (Kleene fixed point theorem)**.** *Let $b \colon P \to P$ be a cocontinuous function on a complete lattice P. Then*

$$\mathsf{gfp}(b) = \bigwedge_{i \in \mathbb{N}} b^i(\top)\,.$$

This theorem gives us an abstract construction for the greatest fixed point of a cocontinuous function $b \colon P \to P$, as follows:

1. $R := \top$;
2. `while` $(R \neq b(R))$ $\{R := b(R)\}$;
3. `return` R;

In case of the function b for a deterministic automaton, as in the previous section, this procedure amounts to an abstract version of the partition refinement algorithm, and it terminates when the state space is finite. Of course, we can also apply it to compute, for instance, the greatest simulation on a deterministic automaton, or another coinductive predicate. It is also interesting to instantiate this procedure to bisimilarity on labelled transition systems: this idea underlies standard algorithms for computing bisimilarity. Both for deterministic automata and transition systems, there are more advanced and efficient versions of partition refinement, but the basic underlying idea is given by Kleene's fixed point theorem.

In general, the above fixed point theorem computes it by approximation, and it can be useful to identify how the greatest fixed point of a given monotone

function looks. Let's see a few more examples. For $\langle o, \delta \colon \rangle X \to 2 \times X^A$ a deterministic automaton, define $s \colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ by

$$s(R) = \{(x, y) \mid o(x) \leq o(y) \text{ and } \forall a \in A.\, (\delta(x)(a), \delta(y)(a)) \in R\}$$

Then it is not hard to check that $s^i(\top) \subseteq X \times X$ relates those pairs of states (x, y) such that all words of length below i accepted by x are also accepted by y.

For $\langle o, t \rangle \colon X \to \mathbb{N} \times X$, defined $\varphi \colon \mathcal{P}(X) \to \mathcal{P}(X)$ by

$$\varphi(P) = \{x \mid o(x) \leq o(t(x)) \text{ and } t(x) \in P\}\,.$$

Then $\varphi^i(\top)$ contains those streams such that the first i elements are non-decreasing, and φ^ω is the intersection of $\{\varphi^i(\top) \mid i \in \mathbb{N}\}$: the non-decreasing streams.

Finally, let $f \colon X \to \mathcal{P}(A \times X)$ be an LTS, and let

$$\psi(P) = \{x \mid \exists (a, x') \in f(x) \text{ s.t. } x' \in P\}\,.$$

Then $\psi^i(\top)$ consists of all states which have a path of length at least i.

In the second part of this lecture, we look at a categorical generalisation of the Kleene fixpoint theorem, to compute final coalgebras. This is not described in these notes; see [1, Section 4.6] for details.

References

[1] B. Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016.