Coalgebra, lecture 10: Algebra and Coalgebra for Regular Expressions

Jurriaan Rot

November 19, 2018

By now we've been talking a lot about coalgebras and coinduction, as well as algebras and induction. It's about time to combine them! As it turns out, this is pretty fundamental: one can use *algebra* to model *syntax/structure*, and *coalgebra* to model *observable behaviour*. As a motivating example, we'll use the good old *regular expressions*. Recognising the combined algebraic/coalgebraic structure gives rise to concrete decision procedures for behavioural equivalence. In this way we can easily check, whether two regular expressions are equivalent. The combined algebraic/coalgebraic structure also allows us to obtain useful proof principles properties about languages in general.

1 Regular expressions

Consider the set 2^{A^*} of languages over A. To speak about regular expressions, we first need some operations on languages:

- union $L + K$, defined by $(L + K)(w) = L(w) \vee K(w)$.
- concatenation $L \cdot K$ (often written as LK), defined by $(L \cdot K)(w) = 1$ iff there are v, u such that $w = vu$ and $L(v) = 1 = K(u)$.
- Kleene star L^*; we first define for a given language L and non-negative integer i, the i-fold composition L^i as $L^0 = 1$ and $L^{i+1} = L \cdot L^i$. The Kleene star is defined as follows: $L^* = \sum_{i \geq 0} L^i$ which means $L^*(w) = 1$ iff there is an i such that $L^i(w) = 1$.

We also have some important constants in the context of regular expressions: the empty language $\emptyset$, the language $\{\varepsilon\}$ just containing the empty word, and the singleton $\{a\}$ for every $a \in A$.

Thus, we're back at the previous lectures: putting all these operations and constants together, we define an algebra on the set 2^{A^*} of languages. First of all, the signature: we have two binary operators (union and concatenation), one

unary (Kleene star) and a bunch of constants. This is modelled by the functor $R\colon \mathsf{Set} \to \mathsf{Set}$, defined by:

$$R(X) = X \times X + X \times X + X + \{0\} + \{1\} + A$$

where from left to right, the components refer to union, concatenation, Kleene star, 0, 1, and $a \in A$ for each a. The corresponding operations on 2^{A^*} then give us an algebra

$$\rho\colon R(2^{A^*}) \to 2^{A^*}$$

thus specifying the 'regular' operations.

In this context, what is a *regular expression*? It is just syntax for the operations and constants that we've defined before. Using our newly found expertise from the previous lectures: they are 'just' the initial algebra of the functor R, that is, the terms over these operations! Let's denote the underlying set of this algebra by Exp_A, so it is an algebra of the form:

$$\alpha\colon R(\mathsf{Exp}_A) \to \mathsf{Exp}_A\,.$$

More explicitly, the set Exp_A is defined by the following grammar:

$$r \;::=\; r + r \mid r \cdot r \mid r^* \mid a \mid 1 \mid 0$$

where a ranges over A. We often abbreviate $r \cdot s$ by rs. For instance, $(a+b)^*$, $a^*(ba^*)^*$, $a+1$, abc, $b(a+ac)^*$ are all examples of regular expressions.

Now, we have our initial algebra α and the algebra ρ defining the regular operations. Thus, we get a unique morphism by intiality:

$$\begin{array}{ccc} R(\mathsf{Exp}_A) & \overset{R(\mathsf{L})}{\dashrightarrow} & R(2^{A^*}) \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\rho} \\ \mathsf{Exp}_A & \underset{\mathsf{L}}{\dashrightarrow} & 2^{A^*} \end{array}$$

If we spell it out, we get:

$$\begin{aligned} \mathsf{L}(0)(w) &= 0 \\ \mathsf{L}(1)(w) &= \begin{cases} 1 & \text{if } w = \varepsilon \\ 0 & \text{otherwise} \end{cases} \\ \mathsf{L}(a)(w) &= \begin{cases} 1 & \text{if } w = a \\ 0 & \text{otherwise} \end{cases} \\ \mathsf{L}(r+s)(w) &= (\mathsf{L}(r) + \mathsf{L}(s))(w) \\ \mathsf{L}(r \cdot s)(w) &= (\mathsf{L}(r)\mathsf{L}(s))(w) \\ \mathsf{L}(r^*) &= \mathsf{L}(r)^* \end{aligned}$$

for all $r, s \in \mathsf{Exp}_A$. The map L is the usual semantics of regular expressions, which we have thus recovered using (initial) algebras!

As usual, for a regular expression r, we call $\mathsf{L}(r)$ the *language denoted by* r, or simply the language of r. A language $L \in 2^{A^*}$ is called *regular* if there is a regular expression r such that $\mathsf{L}(r) = L$.

1.1 The syntactic automaton of regular expressions

In the previous section, we recalled the semantics of regular expressions, and showed how it arises algebraically. As it turns out, we can also retrieve it *coalgebraically*, and the two approaches coincide. Somewhat more precisely, we'll turn the set of all regular expressions into an automaton, whose language semantics coincides with the semantics of expressions.

Before doing so, first recall that we have our beloved final *coalgebra* of the functor $F(X) = 2 \times X^A$ on Set:

$$\langle e, d \rangle \colon 2^{A^*} \to 2 \times (2^{A^*})^A$$

Thus, the set of languages carries both an (interesting) R-algebra and F-coalgebra structure. The connection between these two structures is at the heart of this lecture.

The R-algebra ρ computes the operations on languages. The F-coalgebra $\langle e, d \rangle$ computes the derivatives, and whether the empty word is in the language. The two interact as follows.

Lemma 1.1. *For any two languages L, K and for any $a, b \in A$:*

$$\begin{array}{ll}
\mathsf{0}(\varepsilon) = 0 & \mathsf{0}_a = \mathsf{0} \\
\mathsf{1}(\varepsilon) = 1 & \mathsf{1}_a = \mathsf{0} \\
\{b\}(\varepsilon) = 0 & \{b\}_a = \begin{cases} \mathsf{1} & \textit{if } b = a \\ \mathsf{0} & \textit{otherwise} \end{cases} \\
(L + K)(\varepsilon) = L(\varepsilon) \vee K(\varepsilon) & (L + K)_a = L_a + K_a \\
(L \cdot K)(\varepsilon) = L(\varepsilon) \wedge K(\varepsilon) & (L \cdot K)_a = L_a \cdot K + L(\varepsilon) \cdot K_a \\
L^*(\varepsilon) = \mathsf{1} & (L^*)_a = L_a \cdot L^*
\end{array}$$

In the above, we abuse notation and also use $\mathsf{0}$ to denote $\emptyset$ and $\mathsf{1}$ to denote $\{\varepsilon\}$. Note that in the derivative $(L \cdot K)_a$, we have

$$L_a \cdot K + L(\varepsilon) \cdot K_a = \begin{cases} L_a \cdot K + K_a & \textit{if } L(\varepsilon) = 1 \\ L_a \cdot K & \textit{otherwise} \end{cases}.$$

Proof. Exercise! □

This lemma connects coalgebraic structure (derivatives, output) to algebraic structure (operations). It allow us to compute derivatives of operations inductively: once we know the derivatives of smaller bits, we can derive the derivatives of the entire thing. That's nice, but what is it good for? Well, as we'll see later, one can use this to prove all kinds of properties about languages.

But first, let's return to expressions. Using the above insights, we can now define properly an automaton on the set Exp_A of all expressions, basically by copying the above lemma. This characterisation is due to Brzozowski [1], hence these are often called *Brzozowski derivatives*. For more depth and detail on the (co)algebraic presentation, see [2].

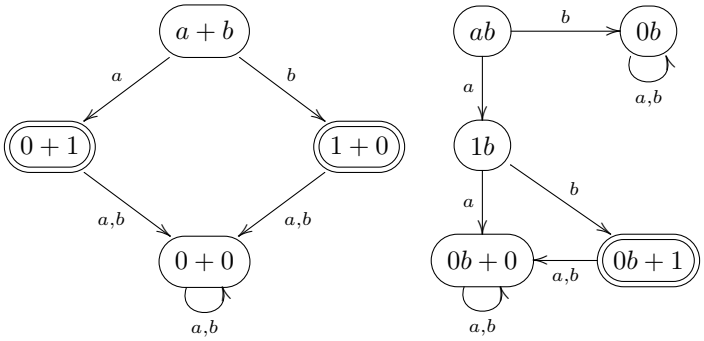
Definition 1.2. We define an automaton $(\mathsf{Exp}_A, \langle o, t\rangle)$, whose state space is the set Exp_A of regular expressions. The functions $o\colon \mathsf{Exp}_A \to 2$ and $t\colon \mathsf{Exp}_A \to \mathsf{Exp}_A^A$ are defined by induction, below. For $r \in \mathsf{Exp}_A$ and $a \in A$, we abbreviate $t(r)(a)$ by r_a.

$$
\begin{array}{ll}
o(\mathsf{0}) = 0 & 0_a = 0 \\
o(\mathsf{1}) = 1 & 1_a = 0 \\
o(b) = 0 & b_a = \begin{cases} 1 & \text{if } b = a \\ 0 & \text{otherwise} \end{cases} \\
o(r + s) = o(r) \vee o(s) & (r + s)_a = r_a + s_a \\
o(r \cdot s) = o(r) \wedge o(s) & (r \cdot s)_a = \begin{cases} r_a \cdot s + s_a & \text{if } o(r) = 1 \\ r_a \cdot s & \text{otherwise} \end{cases} \\
o(r^*) = 1 & (r^*)_a = r_a \cdot r^*
\end{array}
$$

for all $a, b \in A$ and $r, s \in \mathsf{Exp}_A$.

We call (Exp_A, o, t) the *automaton of regular expressions*. For a regular expression $r \in \mathsf{Exp}_A$ and a letter $a \in A$, we call $o(r)$ the *output* and r_a the *a-derivative* of r.

Example 1.3. We draw the part of the automaton (Exp_A, o, t) that starts in state $a + b$ and in ab:



Recall that for every automaton there is a map beh that sends a state to the language it accepts, given by the unique map into the final coalgebra:

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

We have thus obtained two maps of the same type, from regular expressions to languages:

- beh, given by *finality* using the coalgebraic (automaton) structure on Exp_A, and
- L, given by *initiality* using the algebraic structure on 2^{A^*} given by the regular operations of union, concatenation etc.

It is time to connect the two. In a sense, this assures that way we characterised the derivatives of operations is correct.

We are going to prove that the two maps are equal: $\mathsf{beh} = \mathsf{L}$. The idea is to show that L is a coalgebra morphism, from the automaton of regular expressions to the final coalgebra. Since beh is the unique such coalgebra morphism, the result then follows (note that we could also prove that beh is an algebra morphism, but this is harder). So here is the important step:

Lemma 1.4. *The map L is a coalgebra morphism from the automaton of regular expressions to the final coalgebra.*

Proof. We need to prove that

$$o(r) = \mathsf{L}(r)(\varepsilon) \quad \text{and} \quad \mathsf{L}(r)_a = \mathsf{L}(r_a)$$

for all $r \in \mathsf{Exp}_A$, which can be shown by structural induction on r. It essentially follows from Lemma 1.1 and the definition of L.

We only treat the case of concatenation. Suppose it holds for r and s. Then

$$\begin{aligned} o(r \cdot s) &= o(r) \wedge o(s) && \text{(def. } o\text{)} \\ &= \mathsf{L}(r)(\varepsilon) \wedge \mathsf{L}(s)(\varepsilon) && \text{(IH)} \\ &= (\mathsf{L}(r) \cdot \mathsf{L}(s))(\varepsilon) && \text{(Lemma 1.1)} \\ &= \mathsf{L}(r \cdot s) && \text{(def. } \mathsf{L}\text{)} \end{aligned}$$

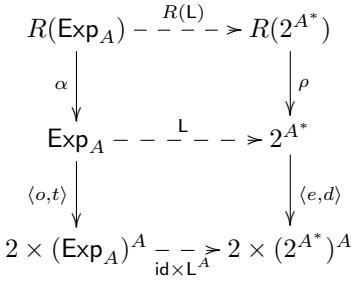
and for all $a \in A$:

$$\begin{aligned} \mathsf{L}(r \cdot s)_a &= (\mathsf{L}(r) \cdot \mathsf{L}(s))_a && \text{(def. } \mathsf{L}\text{)} \\ &= \mathsf{L}(r)_a \cdot \mathsf{L}(s) + \mathsf{L}(r)(\varepsilon) \cdot \mathsf{L}(s)_a && \text{(Lemma 1.1)} \\ &= \mathsf{L}(r_a) \cdot \mathsf{L}(s) + o(r) \cdot \mathsf{L}(s_a) && \text{(IH)} \\ &= \mathsf{L}(r_a \cdot s + o(r) \cdot s_a) && \text{(def. } \mathsf{L}\text{)} \\ &= \mathsf{L}((r \cdot s)_a) && \text{(def. } (r \cdot s)_a\text{)} \end{aligned}$$

Of course, one needs to treat the other cases as well. Anyway, note that all relies on the definition of L, the automaton of regular expressions, and Lemma 1.1. □

Corollary 1.5. *We have $\mathsf{L} = \mathsf{beh}$.*

The situation can be summarised as follows: we have that L is the unique map (both an algebra and a coalgebra morphism!) making the following diagram commute:



1.2 Bisimulations between regular expressions

We have seen that L is defined by initiality, but it is also the coalgebra homomorphism from the automaton of regular expressions to the final coalgebra. This gives us a proof principle: bisimulations! To prove that two regular expressions are language equivalent, it suffices to prove that they are related by bisimilarity $\sim$ on the automaton of regular expressions. Simply instantiating the definition of bisimulations of deterministic automata, we have $r \sim s$ iff there exists a relation $R \subseteq \mathsf{Exp}_A \times \mathsf{Exp}_A$ such that for all $(r, s) \in R$:

1. $o(r) = o(s)$, and
2. $\forall a \in A$: $(r_a, s_a) \in R$.

We summarize the above discussion in the following theorem.

Theorem 1.6. *We have $\mathsf{L}(r) = \mathsf{L}(s)$ iff there exists a bisimulation $R \subseteq \mathsf{Exp}_A \times \mathsf{Exp}_A$ on the automaton of regular expressions such that $(r, s) \in R$.*

For instance, using bisimulations and Theorem 1.6 one can prove that, for any regular expressions r, s, t we have:

1. $\mathsf{L}(r + (s + t)) = \mathsf{L}((r + s) + t)$
2. $\mathsf{L}(r + s) = \mathsf{L}(s + r)$
3. $\mathsf{L}(r + r) = \mathsf{L}(r)$

This is left as an exercise. These equations will play an important role later on.

We would like to apply Theorem 1.6 to define an algorithm that *automatically* proves equivalence of regular expressions, by constructing a bisimulation. To this end, we could try to use the algorithm $\texttt{Naive}(r, s)$. However, the problem is that this algorithm may not terminate! The issue is that, for a regular expression r, the part of the automaton of regular expressions starting in r may be infinite. In other words, r may have infinitely many derivatives.

Example 1.7. Let $A = \{a\}$. Consider the part of the automaton of regular expressions starting in a^*:

$$\overline{a^*} \xrightarrow{a} \overline{1a^*} \xrightarrow{a} \overline{0a^* + 1a^*} \xrightarrow{a} \overline{0a^* + 1a^*} \xrightarrow{a} \overline{0a^* + (0a^* + 1a^*)} \xrightarrow{a} \dots$$

This already goes out of the margin of this document! But of course it's much worse: it goes on forever, the state a^* has infinitely many derivatives. However, notice that each of these derivatives is language equivalent to a^*.

To make this problem more precise, we define the set of derivatives $D(r) \subseteq \mathsf{Exp}_A$ of a regular expression as the least set such that

1. $r \in D(r)$, and
2. for each $s \in D(r)$ and $a \in A$: $s_a \in D(r)$.

The problem is thus that $D(r)$ is infinite. When constructing a bisimulation between r and s, in the worst case it may be that every state of $D(r)$ and $D(s)$ is explored: if one of them is infinite, this means that this approach would not terminate, which means that we do not have a proper algorithm.

1.3 Regular expressions modulo ACI

The solution is to identify regular expressions based on a few equations, by defining a relation $\equiv_0 \subseteq \mathsf{Exp}_A \times \mathsf{Exp}_A$ as the least relation such that for all $r, s, t \in \mathsf{Exp}_A$:

$$(r + s) + t \equiv_0 r + (s + t) \qquad \text{(associativity)}$$
$$r + s \equiv_0 s + r \qquad \text{(commutativity)}$$
$$r + r \equiv_0 r \qquad \text{(idempotence)}$$

and then defining $\equiv$ as the congruence closure of $\equiv_0$: the least eequivalence relation such that $\equiv_0 \subseteq \equiv$ and that:

$$\frac{}{r \equiv r} \qquad \frac{r \equiv s}{s \equiv r} \qquad \frac{r \equiv s \qquad s \equiv t}{r \equiv t}$$

$$\frac{r_1 \equiv s_1 \qquad r_2 \equiv s_2}{r_1 + r_2 \equiv s_1 + s_2} \qquad \frac{r_1 \equiv s_1 \qquad r_2 \equiv s_2}{r_1 \cdot r_2 \equiv s_1 \cdot s_2} \qquad \frac{r \equiv s}{r^* \equiv s^*}$$

We say that two regular expressions r, s are *equivalent modulo ACI* (ACI stands for Associativity, Commutativity and Idempotence) if $r \equiv s$. The reason for introducing this, is that every regular expression has only finitely many derivatives, modulo ACI. To make this claim precise, we let

$$[r] = \{s \in \mathsf{Exp}_A \mid r \equiv s\}$$

be the *equivalence class* of a regular expression r: it is the set of all regular expressions s that are equivalent to r, modulo ACI.

Lemma 1.8. *For every regular expression r, the set*

$$\{[s] \in \mathsf{Exp}_A \mid s \in D(r)\}$$

of equivalence classes (modulo ACI) of derivatives of r is finite.

Example 1.9. Recall from Example 1.7 that a^* has infinitely many derivatives:

$$\overline{a^*} \xrightarrow{a} \overline{1a^*} \xrightarrow{a} \overline{0a^* + 1a^*} \xrightarrow{a} \overline{0a^* + 1a^*} \xrightarrow{a} \overline{0a^* + (0a^* + 1a^*)} \xrightarrow{a} \dots$$

However we have

$$0a^* + (0a^* + 1a^*) \equiv (0a^* + 0a^*) + 1a^* \equiv 0a^* + 1a^*$$

The set

$$\{[s] \in \mathsf{Exp}_A \mid s \in D(a^*)\}$$

from Lemma 1.8 is given by

$$\{[a^*], [1a^*], [0a^* + 1a^*]\}$$

which is indeed finite.

1.4 Bisimulation up to ACI

To exploit the fact that every regular expression has finitely many derivatives modulo ACI (Lemma 1.8), we could consider an automaton where states are equivalence classes of regular expressions, modulo ACI. Here, we take a different approach, based on a new up-to technique. For a relation $R \subseteq \mathsf{Exp}_A \times \mathsf{Exp}_A$, define $\mathsf{aci}(R) \subseteq \mathsf{Exp}_A \times \mathsf{Exp}_A$ as follows:

$$\mathsf{aci}(R) = \{(r, s) \mid r \equiv r' \text{ and } s' \equiv s \text{ for some } (r', s') \in R\}$$

Hence, $\mathsf{aci}(R)$ contains all the pairs of regular expressions that are equivalent modulo ACI to some pair in R. Now a *bisimulation up to ACI* is a relation $R \subseteq \mathsf{Exp}_A \times \mathsf{Exp}_A$ such that for all $(r, s) \in R$:

1. $o(r) = o(s)$, and
2. $\forall a \in A$: $(r_a, s_a) \in \mathsf{aci}(R)$, i.e., there exist $(r'_a, s'_a) \in R$ such that $r_a \equiv r'_a$ and $s'_a \equiv s_a$.

We summarize the above discussion in the following theorem.

Theorem 1.10. *Suppose $R \subseteq \mathsf{Exp}_A \times \mathsf{Exp}_A$ is a bisimulation up to ACI on the automaton of regular expressions. Then for any $(r, s) \in R$, we have $\mathsf{L}(r) = \mathsf{L}(s)$.*

Now we can finally adapt use the usual algorithm for equivalence of states of a DFA, which we've seen before, for equivalence of two regular expressions r_0 and s_0. We call it `RegExpEq`:

RegExpEq(r_0, s_0)

```
(1) R is empty; todo is empty;
(2) insert (r_0, s_0) in todo;
(3) while todo is not empty do
 (3.1) extract (r, s) from todo;
 (3.2) if there is (r', s') ∈ R with r ≡ r' and s' ≡ s then continue;
 (3.3) if o(x') ≠ o(y') then return false;
 (3.4) for all a ∈ A,
         insert (r_a, s_a) in todo;
 (3.5) insert (r, s) in R;
(4) return true;
```

We studied this algorithm for *finite* automata in lecture 6; for the termination, we used that the number of pairs in R is bounded by the size of the state space. Here, we need to be more careful of course: indeed, if we wouldn't use up-to-ACI (as in the standard algorithm) it wouldn't even terminate! However, notice that during the algorithm R will only contain derivatives of r_0, s_0, and a new pair is only added if it isn't already there up to ACI; hence, it follows from Lemma 1.8 that the number of pairs which can be added to R is bounded.

1.5 Kleene algebra

Using coalgebraic techniques, we have arrived at an algorithm for equivalence of regular expressions. There's also a completely different method to reason about equivalence, which I can't resist mentioning in this context. It is (at first sight, at least) purely algebraic. The idea is to provide an *axiomatisation* of equivalence of regular expressions.

A *Kleene algebra* (KA) is a set K together with binary operations $+$ and $\cdot$, a unary operation $*$, and constants 0 and 1 (the same as regular expressions—what a coincidence!), such that the following hold for all $x, y, z \in K$:

$$\begin{aligned}
x + (y + z) &= (x + y) + z \\
x + y &= y + x \\
x + x &= x \\
x + 0 &= x \\
x(yz) &= (xy)z \\
x \cdot 1 &= 1 \cdot x = x \\
x \cdot 0 &= 0 \cdot x = 0 \\
(y + z)x &= yx + zx \\
x(y + z) &= xy + xz \\
x^*x + 1 &= x^* \\
xx^* + 1 &= x^* \\
z + yx \leq x &\to y^*z \leq x \\
z + xy \leq x &\to zy^* \leq x
\end{aligned} \tag{1}$$

where, for all $x, y \in K$, $x \leq y$ iff $x + y = y$.

For instance, the set 2^{A^*} of all languages is a Kleene algebra, if we interpret each of the operations as before. We are going to prove this in a minute. Another example is the set of all *regular* languages. Yet another example of a Kleene algebra is the set of all relations on a set X (how would you define the operations?).

Notice that the terms over a Kleene algebra are just the regular expressions. We write $e \equiv_{\mathsf{KA}} f$ if we can derive equivalence of e and f from the above equations (formally, $\equiv_{\mathsf{KA}}$ is the congruence closure of the relation on regular expressions induced by the equations). Now, the wonderful thing is that

$$e \equiv_{\mathsf{KA}} f \text{ iff } \mathsf{L}(e) = \mathsf{L}(f)$$

that is, two regular expressions represent the same language precisely if they are provably equivalent using the axioms of Kleene algebra. From left to right, this is known as *soundness* (every identity in KA is valid for regular expressions), from right to left this is *completeness* (every identity between regular expressions actually holds in Kleene algebra).

Completeness is a big result, proved by Dexter Kozen [3] and independently by Daniel Krob (in a very different way) in the early nineties, after the problem of giving such an axiomatisation was open for decades. There's a lot more to say about that, but that's all a little outside the scope of the lecture for now. Note that we use the equations of KA to reason about equivalence of regular expressions. But what is possibly even better: we can use the algorithm for language equivalence of regular expressions to check which equations are valid in all Kleene algebras! This is useful, for instance, to prove things about algebras of relations.

As a little side remark: is there any connection with coalgebras at all? Well, yes: Bart Jacobs [2] rephrased completeness as follows. Take the automaton Exp_A of all regular expressions, and quotient it by the equations of Kleene algebra; this yields again an automaton, this time over equivalence classes of provably equivalent expressions. Then completeness amount to the statement that this new coalgebra is final in the category of finite (!) deterministic automata (something to think about on a rainy afternoon).

2 Language equivalence with bisimulations

So we just claimed that the the set of all languages 2^{A^*} forms a Kleene algebra, which means each of the equalities in (1) actually holds when we substitute languages for each of the x, y, z's etc. But how do we prove this? Indeed, how do we prove equations between languages in general?

Well, by now there can only be one answer: bisimulations of course! We start with the following observation, which follows immediately from $(2^{A^*}, \langle e, d \rangle)$ being a final coalgebra.

Theorem 2.1. *Let $(L, K) \in 2^{A^*}$. Then $L = K$ iff there exists a bisimulation R on the final coalgebra $(2^{A^*}, \langle e, d \rangle)$ such that $(L, K) \in R$.*

Thus, to prove $L_0 = K_0$, it suffices to come up with a relation R such that $(L_0, K_0) \in R$ and for all $(L, K) \in R$:

1. $L(\varepsilon) = K(\varepsilon)$ and
2. for all $a \in A$: $(L_a, K_a) \in R$.

Example 2.2. We will prove that $L + 0 = L$ for all L. To this end, consider the relation

$$R = \{(L + 0, L) \mid L \in 2^{A^*}\}$$

We show that R is a bisimulation. First, we have

$$(L + 0)(\varepsilon) = L(\varepsilon) \vee 0(\varepsilon) = L(\varepsilon) \vee 0 = L(\varepsilon)$$

using Lemma 1.1. Then, again using Lemma 1.1, we have for any $a \in A$:

$$\begin{aligned}(L + 0)_a &= L_a + 0_a \\ &= L_a + 0 \\ &R\ L_a\,.\end{aligned}$$

Thus, we have shown that R is a bisimulation. By Theorem 2.1, we obtain $L + 0 = L$ for all languages L.

Example 2.3. We prove that $L \cdot 0 = L$. Consider the relation

$$R = \{(L \cdot 0, 0) \mid L \in 2^{A^*}\}$$

To show that R is a bisimulation, we first consider the output:

$$(L \cdot 0)(\varepsilon) = L(\varepsilon) \wedge 0(\varepsilon) = L(\varepsilon) \wedge 0 = 0 = 0(\varepsilon)$$

again using Lemma 1.1. For any $a \in A$, if $L(\varepsilon) = 0$ then

$$(L \cdot 0)_a = L_a \cdot 0\ R\ 0 = 0_a$$

and if $\varepsilon \in L$, then

$$(L \cdot 0)_a = L_a \cdot 0 + 0_a = L_a \cdot 0 + 0 = L_a \cdot 0\ R\ 0 = 0_a$$

Here we used that $L_a \cdot 0 + 0 = L_a \cdot 0$, which follows from the previous example. We have shown that R is a bisimulation, so that $L \cdot 0 = L$ for every language $L \in 2^{A^*}$.

2.1 Bisimulation up to congruence for regular operations

In this section, we introduce an enhancement of the bisimulation proof method for equality of languages. We first illustrate the need for such an enhancement

with a few examples. Consider the property $LL^* + 1 = L^*$ (one of the KA axioms). In order to prove this identity coinductively, we may try to show that

$$R = \{(LL^* + 1, L^*) \mid L \in 2^{A^*}\}$$

is a bisimulation. Using Lemma 1.1, we may prove that $(LL^* + 1)(\varepsilon) = L^*(\varepsilon)$ for any language $L \in 2^{A^*}$ (exercise). Further, for any $a \in A$:

$$(LL^* + 1)_a = L_a L^* + L(\varepsilon) L_a L^* + 0 = L_a L^* = (L^*)_a$$

where the leftmost and rightmost equality are by Lemma 1.1, and in the second step we use some standard identities. Now we have shown that the derivatives are *equal*; this does not show that R is a bisimulation, since for that, the derivatives need to be *related by* R. The solution, however, is straightforward. If we augment the relation R as follows:

$$R' = R \cup \{(L, L) \mid L \in 2^{A^*}\}$$

then the derivatives of $LL^* + 1$ and L^* are related by R'; and it can be shown that R' is a bisimulation. This solves the problem, but it is arguably somewhat inconvenient that additional work is required to deal with derivatives that are already equal. As another motivating example, we consider the relation

$$R = \{(L^* L + 1, L^*) \mid L \in 2^{A^*}\}.$$

The derivatives are (using Lemma 1.1):

$$(L^* L + 1)_a = L_a L^* L + L_a + 0 = L_a(L^* L + 1) \qquad \text{and} \qquad (L^*)_a = L_a L^*$$

The language $L_a L^*$ can be obtained from $L_a(L^* L + 1)$ by replacing $L^* L + 1$ by L^*, and indeed the latter two languages are related by R. However, since these derivatives are not related *directly* by R, this argument does not show R to be a bisimulation. Extending R to an actual bisimulation is possible but requires a bit of work that one would rather skip.

To deal with examples such as the above in a more natural and easy way, we introduce the notion of *bisimulation up to congruence*. This requires the definition of congruence closure.

Definition 2.4. For a relation $R \subseteq 2^{A^*} \times 2^{A^*}$, define the *congruence closure* $\mathsf{cgr}(R)$ of R as the least relation satisfying the following rules

$$\frac{L\ R\ K}{L\ \mathsf{cgr}(R)\ K} \qquad \frac{}{L\ \mathsf{cgr}(R)\ L} \qquad \frac{L\ \mathsf{cgr}(R)\ K}{K\ \mathsf{cgr}(R)\ L} \qquad \frac{L\ \mathsf{cgr}(R)\ K \qquad K\ \mathsf{cgr}(R)\ M}{L\ \mathsf{cgr}(R)\ M}$$

$$\frac{L_1\ \mathsf{cgr}(R)\ K_1 \qquad L_2\ \mathsf{cgr}(R)\ K_2}{L_1 + L_2\ \mathsf{cgr}(R)\ K_1 + K_2} \qquad \frac{L_1\ \mathsf{cgr}(R)\ K_1 \qquad L_2\ \mathsf{cgr}(R)\ K_2}{L_1 \cdot L_2\ \mathsf{cgr}(R)\ K_1 \cdot K_2} \qquad \frac{L\ \mathsf{cgr}(R)\ K}{L^*\ \mathsf{cgr}(R)\ K^*}$$

The first rule ensures that $R \subseteq \mathsf{cgr}(R)$. The three rules on the right in the first row turn $\mathsf{cgr}(R)$ into an equivalence relation. The three rules on the second row ensure that $\mathsf{cgr}(R)$ is closed under the operations under consideration,

which in particular means that $\mathsf{cgr}(R)$ relates languages obtained by (syntactic) substitution of languages related by R. For example, if $(L^*L + 1, L^*) \in R$, then we can derive from the above rules that $L_a(L^*L + 1)\ \mathsf{cgr}(R)\ L_aL^*$:

$$\dfrac{\dfrac{}{L_a\ \mathsf{cgr}(R)\ L_a} \qquad \dfrac{(L^*L + 1)\ R\ L^*}{(L^*L + 1)\ \mathsf{cgr}(R)\ L^*}}{L_a(L^*L + 1)\ \mathsf{cgr}(R)\ L_aL^*} \tag{2}$$

We use the congruence closure in the definition of bisimulation up to congruence.

Definition 2.5. A relation $R \subseteq 2^{A^*} \times 2^{A^*}$ is a *bisimulation up to congruence* if for any pair $(L, K) \in R$:

1. $L(\varepsilon) = K(\varepsilon)$, and
2. for all $a \in A$: $(L_a, K_a) \in \mathsf{cgr}(R)$.

In a bisimulation up to congruence, the derivatives can be related by the congruence $\mathsf{cgr}(R)$ rather than the relation R itself, and therefore, bisimulations up to congruence may be easier to construct than bisimulations. Indeed, to prove that R is a bisimulation up to congruence, the derivatives can be related by familiar equational reasoning.

A bisimulation up to congruence is, in general, not a bisimulation. However, as we show below, it *represents* one.

Theorem 2.6. *If R is a bisimulation up to congruence then $\mathsf{cgr}(R)$ is a bisimulation.*

Proof. We show that any pair (L, K) in $\mathsf{cgr}(R)$ satisfies the properties

1. $L(\varepsilon) = K(\varepsilon)$ and
2. for any $a \in A$: $L_a\ \mathsf{cgr}(R)\ K_a$

of a bisimulation, by induction on $(L, K) \in \mathsf{cgr}(R)$. This amounts to showing that the set of pairs (L, K) satisfying these properties is closed under the inference rules of Definition 2.4. For the base cases:

1. for the pairs contained in R, the conditions are satisfied by the assumption that R is a bisimulation up to congruence;
2. the case $L\ \mathsf{cgr}(R)\ L$ is trivial.

Now assume languages L, K, M, N such that $L\ \mathsf{cgr}(R)\ K$, $M\ \mathsf{cgr}(R)\ N$, $L(\varepsilon) = K(\varepsilon)$, $M(\varepsilon) = N(\varepsilon)$ and for all $a \in A$: $L_a\ \mathsf{cgr}(R)\ K_a$ and $M_a\ \mathsf{cgr}(R)\ N_a$. We need to prove that $(L + M, K + N), (LM, KN), (L^*, K^*)$, (K, L) and (L, N) (if $K = M$) again satisfy the properties of a bisimulation, i.e., $(L + M)(\varepsilon) = (K + N)(\varepsilon)$ and for all $a \in A$: $(L + M)_a\ \mathsf{cgr}(R)\ (K + N)_a$, and similarly for the other operations. We treat the case of union: $(L + M)(\varepsilon) = L(\varepsilon) \vee M(\varepsilon) =$

$K(\varepsilon) \vee N(\varepsilon) = (K + N)(\varepsilon)$; moreover by assumption and closure of $\mathsf{cgr}(R)$ under $+$ we have $L_a + M_a$ $\mathsf{cgr}(R)$ $K_a + N_a$, and so $(L + M)_a = L_a + M_a$ $\mathsf{cgr}(R)$ $K_a + N_a = (K + N)_a$.

Concatenation and Kleene star are treated in a similar manner, and symmetry and transitivity hold as well (exercise). Thus, by induction, $\mathsf{cgr}(R)$ is a bisimulation, so by Theorem 2.1 we have $L = K$ for any L $\mathsf{cgr}(R)$ K and for any $(L, K) \in R$, in particular. □

This gives us a proof principle:

Corollary 2.7. *If R is a bisimulation up to congruence then for any $(L, K) \in R$, we have $L = K$.*

Any bisimulation is also a bisimulation up to congruence, so Corollary 2.7 is a generalization of Theorem 2.1 for the case of languages. Consequently, its converse holds as well.

We proceed with a number of example proofs based on bisimulation up to congruence.

Example 2.8. Recall the relation

$$R = \{(L^*L + 1, L^*) \mid L \in 2^{A^*}\}$$

from the beginning of this section. As we have seen, the a-derivatives are $L_a(L^*L + 1)$ and L_aL^*, which are not related by R. However, it is shown in (2) that they *are* related by $\mathsf{cgr}(R)$. So R is a bisimulation up to congruence, and consequently $L^*L + 1 = L^*$, by Corollary 2.7.

Moreover, the relation $\{(LL^* + 1, L^*) \mid L \in 2^{A^*}\}$ from the beginning of this section is a bisimulation up to congruence as well; there, the derivatives are equal and thus related by $\mathsf{cgr}(R)$.

Example 2.9. Arden's rule states that if $L = KL + M$ for some languages L, K and M, and K does not contain the empty word, then $L = K^*M$. In order to prove it, let L, K, M be languages such that $K(\varepsilon) = 0$ and $L = KL + M$, and let

$$R = \{(L, K^*M)\}.$$

Using that $K(\varepsilon) = 0$, we have $L(\varepsilon) = (KL + M)(\varepsilon) = (K(\varepsilon) \wedge L(\varepsilon)) \vee M(\varepsilon) = (0 \wedge L(\varepsilon)) \vee M(\varepsilon) = M(\varepsilon) = 1 \wedge M(\varepsilon) = K^*(\varepsilon) \wedge M(\varepsilon) = K^*M(\varepsilon)$. Further,

$$\begin{aligned} L_a = (KL + M)_a &= K_aL + K(\varepsilon) \cdot L_a + M_a \\ &= K_aL + M_a \ \mathsf{cgr}(R) \ K_aK^*M + M_a = (K^*M)_a \end{aligned}$$

for any $a \in A$. So R is a bisimulation up to congruence, proving Arden's rule.

Arden's rule is closely related to the axiom $z + yx \leq x \to y^*z \leq x$ in Kleene algebra. However, the latter do not talk about the empty word. We cover it as well:

Example 2.10. In order to prove $M + KL \subseteq L \Rightarrow K^*M \subseteq L$ (one of the Kleene algebra axioms!) we use that $K^*M \subseteq L$ if and only if $K^*M + L = L$, and try to prove that the relation

$$R = \{(K^*M + L, L) \mid M + KL \subseteq L;\ L, K, M \in 2^{A^*}\}$$

is a bisimulation up to congruence. Let L, K, M be such languages and note that $M + KL + L = L$. Since $(M + KL + L)(\varepsilon) = L(\varepsilon)$ it follows that $(M + L)(\varepsilon) = L(\varepsilon)$, so $(K^*M + L)(\varepsilon) = L(\varepsilon)$. For any alphabet letter a we have

$$\begin{aligned}
(K^*M + L)_a &= K_a K^*M + M_a + L_a \\
&= K_a K^*M + M_a + (M + KL + L)_a \\
&= K_a K^*M + M_a + M_a + K_a L + K(\varepsilon)L_a + L_a \\
&= K_a(K^*M + L) + M_a + K(\varepsilon)L_a + L_a \\
&\ \mathsf{cgr}(R) K_a L + M_a + K(\varepsilon)L_a + L_a \\
&= (M + KL + L)_a \\
&= L_a\,.
\end{aligned}$$

In conclusion, R is a bisimulation up to congruence, proving $M + KL \subseteq L \Rightarrow K^*M + L = L$.

Example 2.11. We prove that for any language L: if $LL = 1$ then $L = 1$. Assume $LL = 1$ and let $R = \{(L, 1)\}$. Since $(LL)(\varepsilon) = 1(\varepsilon) = 1$ also $L(\varepsilon) = 1 = 1(\varepsilon)$. We show that the derivatives of L and 1 are equal, turning R into a bisimulation up to congruence. First, for any $a \in A$:

$$L_a L + L_a = L_a L + L(\varepsilon)L_a = (LL)_a = 1_a = 0\,.$$

Now, one proves that this implies $L_a = 0$ (exercise). Thus $L_a = 0 = 1_a$, so $L_a\ \mathsf{cgr}(R)\ 1_a$.

2.2 Final remarks

The up-to techniques that we considered above, crucially rely on the interplay between algebra and coalgebra: basically, they allows us to use algebraic reasoning (congruence) within the coalgebraic (bisimulation) technique. In the next lecture, we'll study up-to techniques on their own right.

References

[1] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[2] Bart Jacobs. A bialgebraic review of deterministic automata, regular expressions and languages. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José

Meseguer, editors, *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, volume 4060 of *Lecture Notes in Computer Science*, pages 375–404. Springer, 2006.

[3] Dexter Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994.