Coalgebra, lecture 7: Non-deterministic systems

Jurriaan Rot

October 18, 2018

The examples of coalgebras that we've seen so far—streams, automata, trees and variations—are all deterministic. In this lecture we'll study coalgebras with non-determinism, which typically make use of the powerset functor $\mathcal{P}$. In particular, we start out with labelled transition systems, which are a classical example, and a fundamental model of computation which is used a lot, e.g., in concurrency theory and semantics. The associated notion of bisimulation predates, and inspired, the theory of coalgebras. Then we'll consider non-deterministic automata and their determinisation. Based on these techniques, we finally arrive at an efficient algorithm for equivalence of non-determistic automata, using a smart variation of bisimulations.

The part on non-deterministic automata is partially based on lecture notes by Filippo Bonchi and myself (see [1]).

1 Labelled transition systems

A labelled transition system (LTS) over a set of labels A consists of a set X and a transition relation $\to \subseteq X \times A \times X$. We write $x \xrightarrow{a} x'$ for $(x, a, x') \in \to$. For instance:

$$\begin{tikzcd}
x_1 \arrow[r, "a"] & x_2 \arrow[r, bend left, "b"] & x_3 \arrow[l, "a"] & & y_1 \arrow[r, "a"] & y_2 \arrow[r, "b"] & y_3 \arrow[r, bend left, "a"] & y_4 \arrow[l, "b"] \arrow[lll, bend left, "b"]
\end{tikzcd} \tag{1}$$

As always, we'll be concerned with whether states 'behave the same'. For instance, should x_1 and y_1 be deemed equivalent?

They're not isomorphic in any sense: the right-hand side LTS has more states. However, they do generate the same paths of actions, or 'traces'. To make this more precise define *trace semantics* of an LTS is the unique map $\mathsf{tr}\colon X \to 2^{A^*}$ such that $\mathsf{tr}(x)(\varepsilon) = 1$, and $\mathsf{tr}(x)(aw) = 1$ iff $\exists x'. x \xrightarrow{a} x'$ and $\mathsf{tr}(x')(w) = 1$. We say x, y are *trace equivalent* if $\mathsf{tr}(x) = \mathsf{tr}(y)$. For instance, in the above example, $\mathsf{tr}(x_1)$ contains the empty word, and every word of alternating a's and

b's starting with an a. We have that x_1 and y_1 are trace equivalent. But is trace equivalence always the right notion of behavioural equivalence?

Let's consider another example: the famous coffee machine. The problem of modelling coffee machines is a driving force behind the theory of LTSs![1] So, here goes, two machines:

$$\begin{tikzcd}
& u_1 \arrow[d, "\text{coin}"] & \\
& u_2 \arrow[dl, "\text{req-coffee}"] \arrow[dr, "\text{req-tea}"'] & \\
u_3 \arrow[uur, bend left, "\text{coffee}"] & & u_4 \arrow[uul, bend right, "\text{tea}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
& v_1 \arrow[dl, "\text{coin}"'] \arrow[dr, "\text{coin}"] & \\
v_2 \arrow[d, "\text{req-coffee}"'] & & v_3 \arrow[d, "\text{req-tea}"] \\
v_4 \arrow[uur, bend right, "\text{coffee}"'] & & v_5 \arrow[uul, bend left, "\text{tea}"]
\end{tikzcd}$$

Are u_1 and v_1 equivalent? Well, they are certainly trace equivalent. However, these model quite different coffee machines. An 'external observer', that is, user of the coffee machine, can only see the available actions, not the states: so in the beginning one just has the possibility of inserting a coin. In u_1, this takes the machine to u_2; afterwards, the user has the possibility of choosing coffee or tea. In contrast, on the right-hand side, after inserting a coin the user *either* can choose only tea, or only coffee.

The two are distinguished by our favorite notion of equivalence: *bisimulation*. For a labelled transition system $(X, \to)$, a relation $R \subseteq X \times X$ is a bisimulation if for all $(x, y) \in R$:

- if $x \xrightarrow{a} x'$ then $\exists y' \in X$ s.t. $y \xrightarrow{a} y'$ and $(x', y') \in R$, and
- if $y \xrightarrow{a} y'$ then $\exists x' \in X$ s.t. $x \xrightarrow{a} x'$ and $(x', y') \in R$.

As always, the greatest bisimulation is called bisimilarity, and denoted by $\sim$, and states x, y are called bisimilar if $x \sim y$. For instance, in (1), we have $x_1 \sim y_1$: it is easy to check that (x_1, y_2) is contained in a bisimulation. However, the coffee machines are not bisimilar: (u_1, v_1) can not be contained in a bisimulation. For suppose $(u_1, v_1) \in R$ for some bisimulation R; then $v_1 \xrightarrow{\text{coin}} v_2$, to which u_1 can only 'answer' with $u_1 \xrightarrow{\text{coin}} u_2$, so $(u_2, v_2) \in R$. But now $u_2 \xrightarrow{\text{req-coffee}} u_3$, which can't be matched by v_2. So R is not a bisimulation; contradiction.

Bisimulation distinguishes between the two coffee machines, whereas trace equivalence does not. In fact, one can show that bisimilarity implies traces equivalence, which is left as an exercise. But what is so special about bisimilarity? Well, there's several answers to that ... and bisimulation occurs in numerous places (also, for instance, in logic). In any case, it really captures when two states behave the same to an 'external observer', who doesn't have access to the states. Indeed, within the theory of coalgebras, it is the canonical notion of behavioural equivalence, which we'll see next.

[1] Have a look at this wonderful video: it keeps occupying the mind of researchers, and was deeply inspiring for the algorithm we consider later today. `https://cacm.acm.org/magazines/2015/2/182642-hacking-nondeterminism-with-induction-and-coinduction/abstract`

Recall the powerset functor $\mathcal{P}\colon \mathsf{Set} \to \mathsf{Set}$. We define the functor $F\colon \mathsf{Set} \to \mathsf{Set}$ by $F(X) = \mathcal{P}(A \times X)$. An F-coalgebra now is a map $f\colon X \to \mathcal{P}(A \times X)$, which corresponds to an LTS: for every state x, $f(x)$ is the set of transitions from x. And now, a *bisimulation* between F-coalgebras, instantiating the abstract coalgebraic notion, coincides with a bisimulation as defined concretely above! This, as well, we leave as an exercise.

Here are some brief comments on final coalgebras for LTSs. The functor F does not have a final coalgebra, as we've seen in one of the previous lectures (for $\mathcal{P}$). However, we can restrict things a little to obtain one, using the finite powerset functor $\mathcal{P}_f$, given by $\mathcal{P}_f(X) = \{S \subseteq X \mid S \text{ is finite}\}$. Define $F_{fb}\colon \mathsf{Set} \to \mathsf{Set}$ by $F_{fb}(X) = \mathcal{P}_f(A \times X)$. An F_{fb}-coalgebra is called a *finitely branching* labelled transition system. Alternatively, one can take the functor F_{if} given by $F_{if}(X) = (\mathcal{P}_f(X))^A$. An F_{if}-coalgebra is called an *image-finite* LTS (what's the difference between image-finite and finitely branching LTSs?). The functor F_{fb} (and also F_{sb}) does have a final coalgebra: it consists of equivalence classes of rooted trees edge-labelled in A (with multiple, or zero edges possible) w.r.t. bisimilarity. Behavioural equivalence coincides with bisimilarity; and often it's most useful to just consider bisimilarity on the LTS at hand (rather than looking at the image in the final coalgebra).

2 Non-deterministic automata

We've made quite a fuss about deterministic automata (DA). A *non-deterministic* automaton (NDA)[2] is a coalgebra for the functor $F\colon \mathsf{Set} \to \mathsf{Set}$, $F(X) = 2 \times (\mathcal{P}_f(X))^A$. That is, it consists of a set X and a pair

$$\langle o, \delta \rangle \colon X \to 2 \times (\mathcal{P}_f(X))^A$$

of an output function $o\colon X \to 2$ and a transition function $\delta\colon X \to (\mathcal{P}_f(X))^A$. For a state x and label a, we think of $\delta(x)(a)$ as the set of states reached from x with an a-transition. To emphasise this, we sometimes write $x \xrightarrow{a} x'$ for $x' \in \delta(x)(a)$. Note that F is the composition of the functor for deterministic automata and the powerset functor.

The language semantics of a non-deterministic automaton $(X, \langle o, \delta \rangle)$ is given by the map $l\colon X \to 2^{A^*}$, defined for all $x \in X$, $a \in A$ and $w \in A^*$ by:

(a) $l(x)(\varepsilon) = o(x)$,

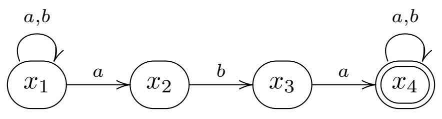
(b) for all $a \in A$ and $w \in A^*$, $l(x)(aw) = 1$ iff there exists $y \in \delta(x)(a)$ such that $l(y)(w) = 1$.

For $b_1, b_2, \ldots, b_n \in 2$, we let $\bigvee_i b_i = b_1 \vee b_2 \vee \ldots \vee b_n$ where $\vee$ is the usual Boolean 'or'. The second point in the above definition can then be conveniently

[2]One typically speaks of an NFA, for nondeterministic *finite* automata. However, we won't require the state space to be finite, for now.

rewritten as $l(x)(aw) = \bigvee_{y \in \delta(x)(a)} l(y)(w)$. We say that $x, y \in X$ are *language equivalent* if $l(x) = l(y)$.

Here's an example of a non-deterministic automaton:



We have $l(x) = \{w \in \{a,b\}^* \mid w \text{ has } aba \text{ as subword}\}$.

The functor F has a final coalgebra; and behavioural equivalence coincides with bisimilarity, so let's have a look at the latter. It is a minor variation on labelled transition systems, and indeed, if we work out the details, it turns out that the coalgebraic definition amounts to: $R \subseteq X \times X$ is a bisimulation if for all $(x, y) \in R$:

- $o(x) = o(y)$,
- if $x \xrightarrow{a} x'$ then $\exists y' \in X$ s.t. $y \xrightarrow{a} y'$ and $(x', y') \in R$, and
- if $y \xrightarrow{a} y'$ then $\exists x' \in X$ s.t. $x \xrightarrow{a} x'$ and $(x', y') \in R$.

This is stronger than language equivalence! It's much the same problem situation as with LTSs: for instance, if we consider the coffee machines as non-deterministic automata (with, say, all of the states accepting) then u_1 and v_1 are language equivalent but they are not bisimilar.

Similarly to LTSs, we have that bisimilarity of NFAs implies language equivalence (exercise), but not the converse, as shown by the coffee machine example. Does this mean the coalgebraic perspective fails here, as bisimilarity (and behavioural equivalence) is too strong? Not quite: it's just that usually, in NFAs, we're interested in traces, and not in bisimilarity (which takes into account the precise moment of branching).

2.1 Determinisation

An early automata theory course typically introduces deterministic automata, non-deterministic automata and then shows that any non-deterministic automata can be 'determinised', preserving the language. But since this is a master course, we'll make this pretty intuitive construction look really complicated! It will be our first example of interplay between coalgebra and algebra, which we'll cover more in-depth in later lectures.

2.1.1 Semi-lattices

So after all this coalgebra, here's an algebraic structure. A *semi-lattice* $(X, +, 0)$ consists of a set X and a function $+\colon X \times X \to X$ which is associative, commutative, idempotent (ACI), and has $0 \in X$ as unit. This means that for all $x, y, z \in X$,

- $x + (y + z) = (x + y) + z$ (associativity)
- $x + y = y + z$ (commutativity)
- $x + x = x$ (idempotence)
- $x + 0 = x$ (unit)

Given two semi-lattices $(X_1, +_1, 0_1)$ and $(X_2, +_2, 0_2)$, a *semi-lattice homomorphism* is a function $f\colon X_1 \to X_2$ such that for all $x, y \in X_1$, $f(x +_1 y) = f(x) +_2 f(y)$ and $f(0_1) = 0_2$.

Example 2.1.

- The set $2 = \{0, 1\}$ is a semi-lattice when taking $+$ to be the ordinary Boolean 'or', often denoted by $\vee$. and 0 is the empty language.
- More generally, for any set S, $\mathcal{P}_f(S)$ is a semi-lattice where $+$ is the union of sets and 0 is the empty set.
- The set of all languages 2^{A^*} is a semi-lattice; for $f, g \in 2^{A^*}$, we let $(f + g)(w) = f(w) \vee g(v)$, and 0 the function which maps everything to $0 \in 2$. Thus, this is essentially union of languages. We sometimes denote $f + g$ by $f \vee g$ as well.
- Generalising the previous example, if $(S, +, 0)$ is a semi-lattice, then S^T is again a semi-lattice, for any set T. For $f, g \in S^T$, we let $(f + g)(x) = f(x) + g(x)$ for all $x \in T$. (What is 0?)
- The function $g\colon \mathcal{P}_f(2) \to 2$ given by $g(\emptyset) = 0$, $g(\{0\}) = 0$, $g(\{1\}) = 1$ and $g(\{0, 1\}) = 1$ is a homomorphism. Let $g'\colon \mathcal{P}_f(2) \to 2$ be defined by $g'(\{0, 1\}) = 0$ and equal to g on the other elements. Then g' is not a homomorphism (why not?).

The following fact is super-important for our purposes. It shows that $\mathcal{P}_f(S)$ is a special semi-lattice.

Lemma 2.2. *Let S be a set, and $(X, +, 0)$ a semi-lattice. For every map $f\colon S \to X$, there exists a unique semi-lattice homomorphism $f^\sharp\colon \mathcal{P}_f(S) \to X$ such that $f^\sharp(\{x\}) = f(x)$ for all $x \in S$.*

This is left as an exercise, but here's the idea. That $f^\sharp$ is a homomorphism means that $f^\sharp(S \cup T) = f^\sharp(S) + f^\sharp(T)$ and $f^\sharp(\emptyset) = 0$. This essentially chacterises $f^\sharp$ everywhere except on the singletons $\{x\}$, where it is given by f.

For instance, taking the identity function $\mathsf{id}\colon 2 \to 2$, we get a unique map $\mathsf{id}^\sharp\colon \mathcal{P}_f(2) \to 2$, which coincides with the map g in Example 2.1.

2.1.2 Determinisation construction

Consider a non-deterministic automaton

$$\langle o, \delta \rangle \colon X \to 2 \times (\mathcal{P}_f(X))^A \,.$$

We define a *deterministic* automaton

$$\langle o^\sharp, \delta^\sharp \rangle \colon \mathcal{P}_f(X) \to 2 \times (\mathcal{P}_f(X))^A \,.$$

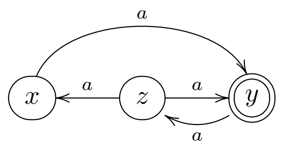
on the set $\mathcal{P}_f(X)$, as follows:

$$o^\sharp(S) = \bigvee_{x \in S} o(x) \tag{2}$$

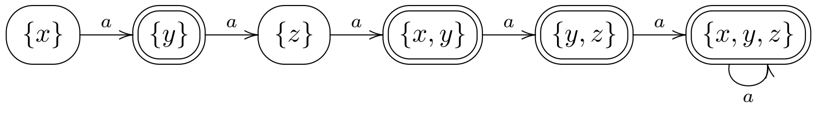
$$\delta^\sharp(S)(a) = \bigcup_{x \in S} \delta(x)(a) \,. \tag{3}$$

Equivalently, $o^\sharp(S) = 1$ iff $\exists x \in S.\, o(x) = 1$, and $y \in \delta^\sharp(S)(a)$ iff $\exists x \in S.\, y \in \delta(x)(a)$. We call $(\mathcal{P}_f(X), \langle o^\sharp, \delta^\sharp \rangle)$ the *determinisation* of $(X, \langle o, \delta \rangle)$. These are precisely the unique semi-lattice homomorphisms obtained by applying Lemma2.2 to o and δ, using that 2 and $(\mathcal{P}_f(X))^A$ are semi-lattices.

Example 2.3. Consider the following non-deterministic automaton:



Part of the determinisation of this automaton (only the stuff starting from $\{x\}$) looks as follows:



As you can see in the above example, and possibly remember from an earlier automata theory course, the language of a state x in the original NDA coincides with the language of $\{x\}$ in the determinisation. This is really the point of the construction, and we will now see how this fits into the current perspective.

First, since $(\mathcal{P}_f(X), \langle o^\sharp, \delta^\sharp \rangle)$ is a deterministic automaton, we obtain a unique map $\mathsf{beh} \colon \mathcal{P}_f(X) \to 2^{A^*}$ to the final coalgebra:

$$\begin{tikzcd}
\mathcal{P}_f(X) \arrow[r, "\mathsf{beh}"] \arrow[d, "\langle o^\sharp, \delta^\sharp \rangle"'] & 2^{A^*} \arrow[d, "\langle e, d \rangle"] \\
2 \times (\mathcal{P}_f(X))^A \arrow[r, "\mathsf{id} \times \mathsf{beh}^A"'] & 2 \times (2^{A^*})^A
\end{tikzcd}$$

Concretely, we have:

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

for all $S \in \mathcal{P}_f(X)$, $a \in A$, $w \in A^*$. Then beh satisfies the following property:

Lemma 2.4. *Let $(\mathcal{P}_f(X), \langle o^\sharp, \delta^\sharp\rangle)$ be the determinisation of an NDA $(X, \langle o, \delta\rangle)$. Then $\mathsf{beh}\colon \mathcal{P}_f(X) \to 2^{A^*}$ is a semi-lattice homomorphism, from the semi-lattice $(\mathcal{P}_f(X), \cup, \emptyset)$ to the semi-lattice 2^{A^*} in Example 2.1 (also given by union).*

This means that:

$$\mathsf{beh}(\{x_1, \ldots, x_n\}) = \mathsf{beh}(\{x_1\}) \cup \ldots \cup \mathsf{beh}(\{x_n\})\,,$$

that is, the language of a set of states in the determinised automaton is given by the language of each of the singletons.

So, taking stock:

- We start out with a non-deterministic automaton $(X, \langle o, \delta\rangle)$. It has a language semantics $l\colon X \to 2^{A^*}$.
- We transform it to a deterministic automaton $(\mathcal{P}_f(X), \langle o^\sharp, \delta^\sharp\rangle)$. This, in turn, has a language semantics $\mathsf{beh}\colon \mathcal{P}_f(X) \to 2^{A^*}$ coming from the final coalgebra.

Now, the point of all this, is that these are the same: the language $l(x)$ of a state $x \in X$ of the original non-deterministic automaton coincides with the language of the singleton $\{x\}$ in the determinized automaton.

Theorem 2.5. *Let $(\mathcal{P}_f(X), \langle o^\sharp, \delta^\sharp\rangle)$ be the determinisation of an NDA $(X, \langle o, \delta\rangle)$. Then for all $x \in X$:*

$$\mathsf{beh}(\{x\}) = l(x)\,.$$

Proof. We prove that for all $w \in A^*$ and for all $x \in X$, $\mathsf{beh}(\{x\}) = l(x)$, by induction on w.

For the base case, we have $\mathsf{beh}(\{x\})(\varepsilon) = o(x) = l(\varepsilon)$. Next, let $w \in A^*$, and suppose for all $x \in X$, $\mathsf{beh}(\{x\}) = l(x)$. Let $a \in A$. Then

$$\begin{aligned}
\mathsf{beh}(\{x\})(aw) &= \mathsf{beh}\left(\bigcup_{y\in\{x\}} \delta(y)(a)\right)(w) && \text{(beh a coalgebra homomorphism)}\\
&= \mathsf{beh}(\delta(x)(a))(w)\\
&= \left(\bigvee_{y\in\delta(x)(a)} \mathsf{beh}(\{y\})\right)(w) && \text{(Lemma 2.4)}\\
&= \bigvee_{y\in\delta(x)(a)} (\mathsf{beh}(\{y\})(w)) && \text{(definition $\vee$ on 2^{A^*})}\\
&= \bigvee_{y\in\delta(x)(a)} (l(y)(w)) && \text{(induction hypothesis)}\\
&= l(x)(aw) && \text{(definition l)}
\end{aligned}$$

This concludes the inductive step. □

2.2 Using bisimulations to compute language equivalence of non-deterministic automata

We've now seen:

- the language semantics of non-deterministic automata, which does *not* coincide with bisimilarity for the associated functor;
- determinisation of a non-deterministic automaton, which yields the language semantics of a state x as the semantics $\mathsf{beh}(\{x\})$ of the singleton.

By the second point, we get something nice, since in the last lecture, we've seen that we can prove behavioural equivalence of deterministic automata using bisimulations! If we instantiate the notion of bisimulation of deterministic automata to a determinised automaton $(\mathcal{P}_f(X), \langle o^\sharp, \delta^\sharp \rangle)$, we get the following. A relation $R \subseteq \mathcal{P}_f(X) \times \mathcal{P}_f(X)$ is a bisimulation if for all $(S,T) \in R$:

- $o^\sharp(S) = o^\sharp(T)$, and
- for all $a \in A$: $(\delta^\sharp(S)(a), \delta^\sharp(T)(a)) \in R$.

By the results of the previous lecture, we know that $\mathsf{beh}(\{x\}) = \mathsf{beh}(\{y\})$ iff $(\{x\},\{y\}) \in R$ for some bisimulation R. So, together with Theorem 2.5, we get:

Corollary 2.6. *Let $(X, \langle o, \delta \rangle)$ be an NDA. For any two states $x, y \in X$, we have $l(x) = l(y)$ if and only if there is a bisimulation $R \subseteq \mathcal{P}_f(X) \times \mathcal{P}_f(X)$ on the determinisation $(\mathcal{P}_f(X), \langle o^\sharp, \delta^\sharp \rangle)$ such that $(\{x\},\{y\}) \in R$.*

So to prove language equivalence of states of an NDA, it suffices to construct a bisimulation on the determinisation.

Example 2.7. Consider the following NDA. For simplicity, we view the automaton below as a single automaton; and we're interested in comparing the language of x and u. (We've already seen the left part in Example 2.3).

$$\begin{tikzcd}
x \arrow[rr, bend left=60, "a"] & z \arrow[l, "a"'] \arrow[r, "a"] & y \arrow[l, bend left, "a"] & u \arrow[rr, bend left=60, "a"] \arrow[r, "a"] & w \arrow[l, bend left, "a"] & v \arrow[l, "a"']
\end{tikzcd}$$

If we determinise these automata, we get (as a part):

$$\begin{tikzcd}
\{x\} \arrow[r, "a"] \arrow[d, dash, dashed] & \{y\} \arrow[r, "a"] \arrow[d, dash, dashed] & \{z\} \arrow[r, "a"] \arrow[d, dash, dashed] & \{x,y\} \arrow[r, "a"] \arrow[d, dash, dashed] & \{y,z\} \arrow[r, "a"] \arrow[dl, dash, dashed] & \{x,y,z\} \arrow[loop below, "a"] \arrow[dll, dash, dashed] \\
\{u\} \arrow[r, "a"] & \{v,w\} \arrow[r, "a"] & \{u,w\} \arrow[r, "a"] & \{u,v,w\} \arrow[loop below, "a"] & &
\end{tikzcd}$$

The dashed lines give a bisimulation between $\{x\}$ and $\{u\}$. Hence, we have shown that $l(x) = l(y)$.

3 HKC and bisimulation up to congruence

In the last lecture, we used bisimulations to develop an algorithm for language equivalence of deterministic automata. We ended the lecture with Hopcroft and Karp's algorithm (HK), which computes a bisimulation up to equivalence rather than a bisimulation, improving the plain bisimulation technique considerably.

In particular, starting from a non-deterministic automaton, we can determinise it and then just feed it to the HK algorithm. In fact, the determinisation can be computed on-the-fly, step by step when we need it. However, it turns out that, in the context of determinised non-deterministic automata as we considered in the last section, one can do even better than bisimulation up to equivalence. This exploits the algebraic (semi-lattice) stucture of the state space of the determinised automaton, and is a prime example of a technique which combines algebra and coalgebra. This algorithm was introduced by Bonchi and Pous [2]—the guys from the video mentioned before!

Consider, for instance, the NDA in Example 2.7, and the corresponding determinised automaton. Intuitively a bisimulation up to congruence does not need to relate $\{x, y\}$ with $\{u, v, w\}$ as soon as $\{x\}$ is already related to $\{u\}$ and $\{y\}$ to $\{v, w\}$. To define it formally, we first need the notion of congruence closure.

Definition 3.1. For a relation $R \subseteq \mathcal{P}_f(X) \times \mathcal{P}_f(X)$, define the congruence closure $\mathsf{cgr}(R)$ of R as the least relation satisfying the following rules.

$$\frac{S\,R\,T}{S \;\mathsf{cgr}(R)\; T} \qquad \frac{}{S \;\mathsf{cgr}(R)\; S} \qquad \frac{S \;\mathsf{cgr}(R)\; T}{T \;\mathsf{cgr}(R)\; S} \qquad \frac{S \;\mathsf{cgr}(R)\; T \qquad T \;\mathsf{cgr}(R)\; U}{S \;\mathsf{cgr}(R)\; U}$$

$$\frac{S_1 \;\mathsf{cgr}(R)\; T_1 \qquad S_2 \;\mathsf{cgr}(R)\; T_2}{(S_1 \cup S_2) \;\mathsf{cgr}(R)\; (T_1 \cup T_2)}$$

The first four rules are the same as for the equivalence closure. The last one closes the relation with respect to set union.

Definition 3.2. Let $(X, \langle o, \delta \rangle)$ be an NDA. A relation $R \subseteq \mathcal{P}_f(X) \times \mathcal{P}_f(X)$ is a *bisimulation up to congruence* (on the determinised automaton) iff for all $(S, T) \in R$:

1. $o^\sharp(S) = o^\sharp(T)$ and,
2. for all $a \in A$, $(\delta^\sharp(S)(a), \delta^\sharp(T)(a)) \in \mathsf{cgr}(R)$.

Example 3.3. Recall the NDA from Example 2.7. The relation R given by the dashed lines is a bisimulation up to congruence.

$$\begin{tikzcd}
\{x\} \arrow[r, "a"] \arrow[d, dash, dashed] & \{y\} \arrow[r, "a"] \arrow[d, dash, dashed] & \{z\} \arrow[r, "a"] \arrow[d, dash, dashed] & \{x,y\} \arrow[r, "a"] & \{y,z\} \arrow[r, "a"] & \{x,y,z\} \arrow[loop below, "a"] \\
\{u\} \arrow[r, "a"] & \{v,w\} \arrow[r, "a"] & \{u,w\} \arrow[r, "a"] & \{u,v,w\} \arrow[loop below, "a"] & &
\end{tikzcd}$$

To see this, notice that we have:

$$\dfrac{\dfrac{\{x\}\ R\ \{u\}}{\{x\}\ \mathsf{cgr}(R)\ \{u\}} \qquad \dfrac{\{y\}\ R\ \{v,w\}}{\{y\}\ \mathsf{cgr}(R)\ \{v,w\}}}{\{x,y\}\ \mathsf{cgr}(R)\ \{u,v,w\}}$$

Note that R is *not* a bisimulation.

Bisimulations up to congruence suffice to prove language equivalence. This follows from the following result:

Theorem 3.4. *Let $(X, \langle o, \delta \rangle)$ be an NDA. If $R \subseteq \mathcal{P}_f(X) \times \mathcal{P}_f(X)$ is a bisimulation up to congruence on the determinisation of $(X, \langle o, \delta \rangle)$, then* $\mathsf{cgr}(R)$ *is a bisimulation (on the determinisation).*

In one of the later lectures, we'll consider bisimulations up-to, in general, and give a much nicer proof of the above fact. But, for completeness, here goes.

Proof. Consider the relation

$$\Psi = \{(S,T) \mid o^\sharp(S) = o^\sharp(T) \text{ and } \forall a \in A.\, (\delta^\sharp(S)(a), \delta^\sharp(T)(a)) \in \mathsf{cgr}(R)\}\,.$$

We need to prove that $\mathsf{cgr}(S) \subseteq \Psi$. Since $\mathsf{cgr}(S)$ is characterised as the least relation satisfying the rules in Definition 3.1, it suffices to show that S satisfies all these rules:

- If $(S,T) \in R$, then $(S,T) \in \Psi$ by the assumption that R is a bisimulation up to congruence;
- For all $S \in \mathcal{P}_f(X)$, it is easy to show that $(S,S) \in \Psi$;
- If $(S,T) \in \Psi$ then $(T,S) \in \Psi$ (exercise);
- If $(S,T) \in \Psi$ and $(T,U) \in \Psi$ then $(S,U) \in \Psi$ (exercise);
- Suppose $(S_1,T_1) \in \Psi$ and $(S_2,T_2) \in \Psi$. Then

$$\begin{aligned} o^\sharp(S_1 \cup S_2) &= o^\sharp(S_1) \vee o^\sharp(S_2) && (\text{ since } o^\sharp \text{ a semilattice homomorphism}) \\ &= o^\sharp(T_1) \vee o^\sharp(T_2) && (\text{ since } (S_1,T_1) \in \Psi \text{ and } (S_2,T_2) \in \Psi) \\ &= o^\sharp(T_1 \cup T_2)\,. && (\text{ since } o^\sharp \text{ a semilattice homomorphism}) \end{aligned}$$

Moreover, for all $a \in A$, we have:

$$\begin{aligned} \delta^\sharp(S_1 \cup S_2)(a) &= \delta^\sharp(S_1)(a) \cup \delta^\sharp(S_2)(a) \\ &\qquad\qquad (\text{ since } \delta^\sharp \text{ a semilattice homomorphism}) \\ &\ \mathsf{cgr}(R)\ \delta^\sharp(T_1)(a) \cup \delta^\sharp(T_2)(a) \\ &\qquad\qquad (\text{ since } (S_1,T_1) \in \Psi \text{ and } (S_2,T_2) \in \Psi) \\ &= \delta^\sharp(T_1 \cup T_2) \quad (\text{ since } \delta^\sharp \text{ a semilattice homomorphism}) \end{aligned}$$

This concludes the proof that $\mathsf{cgr}(R) \subseteq \Psi$, and hence that $\mathsf{cgr}(R)$ is a bisimulation. □

Corollary 3.5. *Let $(X, \langle o, \delta \rangle)$ be an NDA. If $R \subseteq \mathcal{P}_f(X) \times \mathcal{P}_f(X)$ is a bisimulation up to congruence on the determinisation of $(X, \langle o, \delta \rangle)$, then for all $x, y \in X$: $(\{x\}, \{y\}) \in R$ implies $l(x) = l(y)$.*

Proof. Suppose R is such a bisimulation. By Theorem 3.4, $\mathsf{cgr}(R)$ is a bisimulation. Let $(\{x\}, \{y\}) \in R$. Then $(\{x\}, \{y\}) \in \mathsf{cgr}(R)$, hence $l(x) = l(y)$ by Corollary 2.6. □

Thus, bisimulations up to congruence suffice to prove language equivalence. And they can be (much) smaller than plain bisimulations!

To conclude, we adapt the HK algorithm to use bisimulation up to congruence, as follows. It takes as input a non-deterministic automaton $(X, \langle o, \delta \rangle)$, and two sets of states $S_0, T_0 \subseteq X$.

```
    HKC($S_0$,$T_0$)

(1) $R$ is empty; $todo$ is empty;
(2) insert $(S_0, T_0)$ in $todo$;
(3) while $todo$ is not empty do
 (3.1)   extract $(S,T)$ from $todo$;
 (3.2)   if $(S,T) \in \mathsf{cgr}(R)$ then continue;
 (3.3)   if $o^\sharp(S) \neq o^\sharp(T)$ then return $false$;
 (3.4)   for all $a \in A$,
           insert $(\delta^\sharp(S)(a), \delta^\sharp(T)(a))$ in $todo$;
 (3.5)   insert $(S,T)$ in $R$;
(4) return $true$;
```

If we instantiate the algorithm with singletons $\{x\}, \{y\}$ the algorithm checks language equivalence, i.e., whether $l(x) = l(y)$.

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.