Proving Language Inclusion and Equivalence by Coinduction

Jurriaan Rot[d,1], Marcello Bonsangue[a,b], Jan Rutten[b,c]

[a]*LIACS – Leiden University, Niels Bohrweg 1, Leiden, Netherlands*
[b]*Centrum Wiskunde en Informatica, Science Park 123, Amsterdam, Netherlands*
[c]*ICIS – Radboud University Nijmegen, Heyendaalseweg 135, Nijmegen, Netherlands*
[d]*Université de Lyon, CNRS, ENS de Lyon, UCBL, LIP, 46 Allée d'Italie, Lyon, France*

Abstract

Language equivalence and inclusion can be checked coinductively by establishing a (bi)simulation on suitable deterministic automata. In this paper we present an enhancement of this technique called *(bi)simulation-up-to*. We give general conditions on language operations for which bisimulation-up-to is sound. These results are illustrated by a large number of examples, giving new proofs of classical results such as Arden's rule, and involving the regular operations of union, concatenation and Kleene star as well as language equations with complement and intersection, and shuffle (closure).

1. Introduction

The set of all languages over a given alphabet can be turned into an (infinite) deterministic automaton. By the *coinduction* principle, any two languages that are *bisimilar* as states in this automaton are equal. The typical way to show that two languages x and y are bisimilar is by exhibiting a *bisimulation*, a relation on languages satisfying certain properties, which contains the pair (x, y). Indeed, this is the basis of a practical coinductive proof method for language equality [30], which has, for example, been applied in effective procedures for checking equivalence of regular languages [8, 19, 21, 30].

In this paper we present *bisimulation up to congruence*, in the context of languages and automata. This is an enhancement of bisimulation originally stemming from process theory [27, 33]. In order to prove bisimilarity of two

*Corresponding author.
Email addresses: `jurriaan.rot@ens-lyon.fr` (Jurriaan Rot), `marcello@liacs.nl` (Marcello Bonsangue), `jan.rutten@cwi.nl` (Jan Rutten)

[1]The research of this author has been funded by the Netherlands Organisation for Scientific Research (NWO), CoRE project, dossier number: 612.063.920. The first author is supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

languages, instead of showing that they are related by a bisimulation, one can show that they are related by a *bisimulation-up-to*, which in many cases yields smaller, easier and more elegant proofs. As such, we introduce a proof method which improves on the more classical coinductive approach based on bisimulations.

Bisimulation-up-to(-congruence) techniques essentially make use of the underlying (algebraic) structure induced on (the automaton of) languages by the operations and expressions under consideration. In this paper we will first focus on languages presented by the regular operations of union, concatenation and Kleene star. We will exemplify our coinductive proof method based on bisimulation-up-to by novel proofs of several classical results such as Arden's rule and the soundness of the axioms of Kleene algebra. This introduces the main ingredients, and the practical use of the proof technique.

Our aim however is to deal with a wide variety of operations on languages. To this end we introduce a general format of *behavioural differential equations*, based on a similar format introduced for streams [20, 32] (called *stream GSOS* in [14]). In the current paper we prove that for any operator defined in this format, the associated bisimulation-up-to techniques are sound. We then apply these results by considering language equations involving intersection and complement, and show the usefulness and versatility of the techniques by giving a full coinductive proof of the fact that two particular context-free languages defined in terms of language equations, that of palindromes and that of non-palindromes, indeed form each others complement. Moreover we give a number of example proofs for the operations of shuffle and shuffle closure.

While bisimilarity can be used to prove equality, the notion of *similarity* can be used to prove language *inclusion*. We will introduce *simulation-up-to* techniques, and show that these are sound whenever the operations under consideration adhere to the above format of behavioural differential equations, and additionally satisfy a monotonicity condition. While language inclusion $x \subseteq y$ can of course be reduced to equality $x + y = y$, it turns out that in practical cases it can be much more convenient and efficient to use simulation-up-to directly, instead of using this reduction and prove equality by bisimulation-up-to.

Behavioural differential equations have been studied extensively, mostly in the context of streams [14, 20, 32]. The format for language operations which we introduce in this paper, is an extension of the format for stream operations introduced in [14, 20]. In [20] it is shown that the operations which can be given by behavioural differential equations are precisely the *causal* functions. For an operation f on streams, f is causal if for any $n > 0$: the first n elements of $f(\sigma_1, \ldots, \sigma_n)$ depend only on the first n elements of each argument $\sigma_1, \ldots, \sigma_n$. This notion has a natural counterpart for languages, and we show that indeed the correspondence holds in our case as well. This gives an additional *semantic* characterization of a large class of operations for which bisimulation-up-to techniques are sound. A similar result is in [9], where causality is taken as a sufficient condition for soundness of up-to techniques for streams.

The main contribution of this paper is the presentation of the coinductive proof technique of (bi)simulation-up-to in the context of languages and automata, together with a large number of examples, providing an accessible explanation of these techniques requiring little background knowledge from the reader. An earlier version of this work appeared as a conference paper [29]. With respect to [29] we have the following new contributions. First, the present paper is entirely self-contained, in contrast to [29] which relies on the abstract theory of coalgebraic bisimulation-up-to [28]. Second, we add here a number of new examples, for other operations such as shuffle. Third, we introduce the notion of *simulation-up-to.* Finally the result stating that operations adhering to the format of behavioural differential equations are precisely the causal functions, is new in the context of languages.

The outline of this paper is as follows. In Section 2 we recall the notions of bisimulation and coinduction in the context of languages and automata. Then in Section 3 we present bisimulation-up-to for the regular operations. In Section 4 we discuss bisimulation-up-to for other operations, by introducing a format for which bisimulation-up-to is guaranteed to be sound and providing a number of examples. In Section 5 we introduce simulation-up-to techniques for language inclusion. In Section 6 we prove the correspondence between behavioural differential equations and causal functions. In Section 7 we place our work in the context of coalgebraic theory and discuss related work, and finally in Section 8 we conclude.

2. Languages, automata, bisimulations and coinduction

Throughout this paper we assume a fixed alphabet A, which is simply a (possibly infinite) set. We denote by A^* the set of *words*, i.e., finite concatenations of elements of A; we denote the empty word by ε, and the concatenation of two words w and v by wv. The set of *languages* over A is given by $\mathcal{P}(A^*)$, and ranged over by x, y, z. We denote the empty language by 0 and the language $\{\varepsilon\}$ by 1. Moreover when no confusion is likely to arise, we write a to denote the language $\{a\}$, for alphabet letters $a \in A$.

A *(deterministic) automaton* over A is a triple (S, o, δ) where S is a set of states, $o\colon S \to \{0, 1\}$ is an output function, and $\delta\colon S \times A \to S$ is a transition function. Notice that S is not necessarily finite, and there is no initial state. We say a state $s \in S$ is *final* or *accepting* if $o(s) = 1$. For each automaton (S, o, δ) there is a function $l\colon S \to \mathcal{P}(A^*)$ which assigns to each state $s \in S$ a language, inductively defined as follows:

$$\varepsilon \in l(s) \text{ iff } o(s) = 1 \qquad\qquad aw \in l(s) \text{ iff } w \in l(\delta(s, a))$$

The classical definition of *bisimulation* [22, 26] applies to labelled transition systems, which, in contrast to deterministic automata, do not feature output and may have a non-deterministic branching behaviour[2]. We will base ourselves

[2]In fact, the standard reference [26] introduces bisimulations for automata rather than transition systems, and Theorem 2.2 appears already there.

on a different notion of bisimulation specific to deterministic automata, which is in fact an instantiation of the general *coalgebraic* definition of bisimulation (see Section 7).

Definition 2.1. Let (S, o, δ) be a deterministic automaton. A *bisimulation* is a relation $R \subseteq S \times S$ such that for any $(s, t) \in R$:

1. $o(s) = o(t)$, and
2. for all $a \in A$: $(\delta(s, a), \delta(t, a)) \in R$.

Given a deterministic automaton, the union of all bisimulations is again a bisimulation, is denoted by $\sim$ and is called *bisimilarity*[3]; if $s \sim t$ for two states s, t then we say these states are *bisimilar*. Notice that in order to show that two states s and t are bisimilar, it suffices to construct a bisimulation R such that $(s, t) \in R$. As it turns out, this gives a proof principle for showing language equivalence of states:

Theorem 2.2 (Coinduction)**.** *For any two states s, t of a deterministic automaton: if $s \sim t$ then $l(s) = l(t)$.*

Proof. By induction on the length of words. For any states s, t such that $s \sim t$ we have $o(s) = o(t)$; so for the empty word, we have $\varepsilon \in l(s)$ iff $o(s) = 1$ iff $o(t) = 1$ iff $\varepsilon \in l(t)$. Next suppose that for any word w of length n and any states s, t: if $s \sim t$, then $w \in l(s)$ iff $w \in l(t)$. Let w be such a word and s, t states such that $s \sim t$; then for any alphabet letter a: $\delta(s, a) \sim \delta(t, a)$. So by assumption $w \in l(\delta(s, a))$ iff $w \in l(\delta(t, a))$, and thus $aw \in l(s)$ iff $w \in l(\delta(s, a))$ iff $w \in l(\delta(t, a))$ iff $aw \in l(t)$. $\square$

Because the automata considered here are deterministic, the converse of the above coinduction principle holds as well, i.e., if $l(s) = l(t)$ then s is related to t by some bisimulation R. Bisimulations R may well be infinite, but this is not necessarily a problem; in practice one can often give a finite description of such an infinite relation.

Next, we recall how bisimulations can be used to establish equality of languages. To do so we recall the notion of *language derivatives*: the a-derivative of a language x is defined as

$$x_a = \{w \mid aw \in x\} .$$

The set $\mathcal{P}(A^*)$ of all languages can be turned into a deterministic automaton by defining the output function and the transition function as follows:

$$o(x) = \begin{cases} 1 & \text{if } \varepsilon \in x \\ 0 & \text{otherwise} \end{cases} \qquad \delta(x, a) = x_a \text{ for all } a \in A .$$

[3]Bisimilarity coincides with the well-known Myhill–Nerode equivalence, and thus factorization of an automaton by the associated bisimilarity relation corresponds to minimization.

One can check that for any language x, the language accepted by the corresponding state in the automaton is precisely x itself.

Spelling out Definition 2.1, a relation R on languages is a bisimulation on this automaton if for any $(x, y) \in R$:

$$o(x) = o(y) \qquad \text{and} \qquad (x_a, y_a) \in R \text{ for any } a \in A\,.$$

In the remainder of this paper, we will only consider bisimulations on this automaton. By the coinduction principle (Theorem 2.2), we have the following method for checking equality of languages x and y: if we can establish a bisimulation on the above automaton containing the pair (x, y), then $x \sim y$, so $x = y$.

2.1. Regular operations

We will be interested in the regular operations on languages, defined in a standard way: union $x + y = \{w \mid w \in x \text{ or } w \in y\}$, concatenation $x \cdot y = \{w \mid w = uv \text{ for some } u \in x \text{ and } v \in y\}$ and Kleene star $x^* = \sum_{i\geq 0} x^i$, where $x^0 = 1$ and $x^{i+1} = x \cdot x^i$. We often write xy for $x \cdot y$.

In order to prove equivalence of languages defined using the above operations we may use bisimulations, but for this we need a characterization of the output (acceptance of the empty word) and the derivatives of languages. Such a characterization was given for regular expressions by Brzozowski [5]; we formulate this in terms of languages (e.g., [7, p. 41]):

Lemma 2.3. *For any two languages x, y and for any $a, b \in A$:*

$$\begin{array}{ll}
0_a = 0 & o(0) = 0 \\
1_a = 0 & o(1) = 1 \\
b_a = \begin{cases} 1 & \text{if } b = a \\ 0 & \text{otherwise} \end{cases} & o(b) = 0 \\
(x+y)_a = x_a + y_a & o(x+y) = o(x) \vee o(y) \\
(x \cdot y)_a = x_a \cdot y + o(x) \cdot y_a & o(x \cdot y) = o(x) \wedge o(y) \\
(x^*)_a = x_a \cdot x^* & o(x^*) = 1
\end{array}$$

Example 2.4. Let us prove that $(a+b)^* = (a^*b^*)^*$ for some alphabet letters a, b (for simplicity we assume that the alphabet does not contain any other letters). To this end, we start with the relation $R = \{((a+b)^*, (a^*b^*)^*)\}$ and try to show it is a bisimulation. So we must show that the outputs of $(a + b)^*$ and $(a^*b^*)^*$ coincide, and that their a-derivatives and their b-derivatives are related by R. Using Lemma 2.3, we see that $o((a + b)^*) = 1 = o((a^*b^*)^*)$. Moreover, again using Lemma 2.3, we have $((a + b)^*)_a = (a + b)_a(a + b)^* = (1 + 0)(a + b)^* = (a + b)^*$ and $((a^*b^*)^*)_a = (a^*b^*)_a(a^*b^*)^* = ((a^*)_ab^* + o(a^*)(b^*)_a)(a^*b^*)^* = (a^*b^* + 0)(a^*b^*)^* = (a^*b^*)^*$, so the a-derivatives are again related (notice that apart from Lemma 2.3, we have used some basic facts about the regular operations). The b-derivative of $(a + b)^*$ is $(a + b)^*$ itself; the b-derivative of $(a^*b^*)^*$ is $b^*(a^*b^*)^*$. But $b^*(a^*b^*)^*$ is equal to $(a^*b^*)^*$, so we are done. For an alternative proof that does not use the latter equality, consider the relation

$R' = R \cup \{((a+b)^*, b^*(a^*b^*)^*)\}$. As it turns out, the pair $((a+b)^*, b^*(a^*b^*)^*)$ satisfies the necessary conditions as well, turning R' into a bisimulation. We leave the details as an exercise for the reader, and conclude $(a+b)^* = (a^*b^*)^*$ by coinduction.

Bisimulation proofs in general will follow the above pattern of using Lemma 2.3 to compute outputs and to expand the derivatives, and then using some reasoning to show that the outputs are equal and the derivatives related. In the sequel we will sometimes use Lemma 2.3 without further reference to it. We note that the above coinductive proof method applies to general languages, not only to regular ones. However, if one restricts to regular languages, then this technique gives rise to an effective algorithm for checking equivalence.

The axioms of *Kleene algebra (KA)* [18] constitute a complete axiomatization of language equivalence of regular expressions. We recall them here for the following two reasons. First, they provide a number of interesting examples for our methods. Second, the axioms (especially (1) through (9)) are often quite useful for relating derivatives. We state the axioms in terms of languages and our concrete operations:

$$x + (y + z) = (x + y) + z \tag{1}$$
$$x + y = y + x \tag{2}$$
$$x + x = x \tag{3}$$
$$x + 0 = x \tag{4}$$
$$x(yz) = (xy)z \tag{5}$$
$$x \cdot 1 = 1 \cdot x = x \tag{6}$$
$$x \cdot 0 = 0 \cdot x = 0 \tag{7}$$
$$(y + z)x = yx + zx \tag{8}$$
$$x(y + z) = xy + xz \tag{9}$$
$$x^*x + 1 = x^* \tag{10}$$
$$xx^* + 1 = x^* \tag{11}$$
$$z + yx \subseteq x \to y^*z \subseteq x \tag{12}$$
$$z + xy \subseteq x \to zy^* \subseteq x \tag{13}$$

Notice that $x \subseteq y$ iff $x + y = y$. All of (1) through (9) follow easily from the definition of the operations. The remaining axioms will be treated below as examples of coinductive proofs.

3. Bisimulation-up-to for regular operations

In this section we will introduce an enhancement of the bisimulation proof method. We first illustrate the need for such an enhancement with an example. Consider the Kleene algebra axiom (11) (see Section 2). In order to prove this identity coinductively, consider the relation $R = \{(xx^* + 1, x^*) \mid x \in \mathcal{P}(A^*)\}$;

let us see if this is a bisimulation. Using Lemma 2.3, it is easy to show that for any language x, the outputs of $xx^* + 1$ and x^* are equal. For any $a \in A$:

$$(xx^* + 1)_a = x_a x^* + o(x) x_a x^* + 0 = x_a x^* = (x^*)_a$$

where the leftmost and rightmost equality are by Lemma 2.3, and in the second step we use some of the KA identities which we know to hold. Now we have shown that the derivatives are *equal*; this does not allow us to conclude that R is a bisimulation, since for that, the derivatives need to be *related by* R. Instead we can consider the relation $R' = R \cup \{(y, y) \mid y \in \mathcal{P}(A^*)\}$. Then the derivatives of $xx^* + 1$ and x^* are related by R'; moreover, the diagonal is easily seen to satisfy the properties of a bisimulation as well. This solves the problem, but is arguably somewhat inconvenient.

Now consider the relation $R = \{(x^*x + 1, x^*) \mid x \in \mathcal{P}(A^*)\}$ which we may try to use to prove (10) by coinduction. The derivatives are (using Lemma 2.3):

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

Clearly $x_a x^*$ can be obtained from $x_a(x^*x + 1)$ by substituting $x^*x + 1$ for x^*, and indeed the latter two languages are related by R. However, unfortunately these derivatives are not related *directly* by R, and so R is not a bisimulation. Extending R to a bisimulation is indeed a non-trivial task.

When proving identities over languages coinductively, situations such as in the above examples occur very often. In fact, similar phenomena occur and have been studied long before in the setting of process algebra; there, *bisimulation-up-to* techniques are used to reduce the size of a bisimulation relation while preserving soundness [27, 33]. Here, we follow a similar development for deterministic automata. To define bisimulation-up-to, we need the notion of congruence closure.

Definition 3.1. For a relation $R \subseteq \mathcal{P}(A^*) \times \mathcal{P}(A^*)$, define the *congruence closure* of R (with respect to $+$, $\cdot$ and *) as the least relation $\equiv$ satisfying the following rules:

$$\frac{x \, R \, y}{x \equiv y} \qquad \frac{}{x \equiv x} \qquad \frac{x \equiv y}{y \equiv x} \qquad \frac{x \equiv y \quad y \equiv z}{x \equiv z}$$

$$\frac{x_1 \equiv y_1 \quad x_2 \equiv y_2}{x_1 + x_2 \equiv y_1 + y_2} \qquad \frac{x_1 \equiv y_1 \quad x_2 \equiv y_2}{x_1 \cdot x_2 \equiv y_1 \cdot y_2} \qquad \frac{x \equiv y}{x^* \equiv y^*}$$

In the sequel we denote the congruence closure of a given relation R by $\equiv_R$, or simply by $\equiv$ if R is clear from the context.

The upper left rule ensures $R \subseteq \equiv_R$. The three rules on the right in the first row ensure that $\equiv_R$ is an equivalence relation. Notice that the reflexivity rule has as a consequence that languages which are equal, are also related by $\equiv_R$. The transitivity rule allows to relate languages in multiple "proof steps". Finally the three rules on the second row ensure that $\equiv_R$ is a congruence, which in particular means that $\equiv_R$ relates languages which are obtained by (syntactic) substitution of languages related by R. For example, if $x^*x + 1 \; R \; x^*$, then we can derive from the above rules that $x_a(x^*x + 1) \equiv_R x_a x^*$.

Definition 3.2. Let $R \subseteq \mathcal{P}(A^*) \times \mathcal{P}(A^*)$ be a relation on languages. We say R is a *bisimulation up to congruence*, or simply a *bisimulation-up-to*, if for any pair $(x, y) \in R$:

1. $o(x) = o(y)$, and
2. for all $a \in A$: $x_a \equiv_R y_a$.

One clear difference with the definition of bisimulation, is that bisimulation-up-to here is only defined on the automaton of languages. The reason is that it uses the *algebraic* structure, i.e., the operations on languages that we consider here.

In a bisimulation-up-to, the derivatives can be related by the *congruence* $\equiv_R$ rather than the relation R itself. Indeed, to prove that R is a bisimulation-up-to, the derivatives can be related by familiar equational reasoning.

Of course, a bisimulation-up-to is, in general, not a bisimulation. However, it *represents* a bisimulation, in the following sense: if R is a bisimulation-up-to, then $\equiv_R$ is a bisimulation. This means that in that case for any $(x, y) \in \equiv_R$ we have $x = y$ by coinduction. Since $R \subseteq \equiv_R$ this holds in particular for all pairs related by R. So we have the following proof principle:

Theorem 3.3 (Coinduction-up-to)**.** *If R is a bisimulation-up-to then for any $(x, y) \in R$: $x = y$.*

Proof. Let $\equiv$ be the congruence closure of R. We show by structural induction that any pair (x, y) in $\equiv$ satisfies the properties of a bisimulation, i.e., $o(x) = o(y)$ and $x_a \equiv y_a$ for any $a \in A$. This essentially amounts to showing that $\equiv$ is closed under the inference rules of Definition 3.2. 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;
2. the case $x \equiv x$ is trivial.

Suppose pairs of languages $(x, y), (u, v) \in \equiv$ satisfy the desired properties. We need to prove that $(x + u, y + v), (xu, yv), (x^*, y^*)$ (regular operators), (y, x) (symmetry) and (x, v) (transitivity, only if $y = u$) again satisfy the properties of a bisimulation. We treat the case of union: $o(x + u) = o(x) \vee o(u) = o(y) \vee o(v) = o(y + v)$; moreover by assumption and closure of $\equiv$ under $+$ we have $x_a + u_a \equiv y_a + v_a$, and so $(x + u)_a = x_a + u_a \equiv y_a + v_a = (y + v)_a$.

Concatenation and Kleene star can be treated in a similar manner, and symmetry and transitivity are not difficult either. Thus by induction, $\equiv$ is a bisimulation; so by coinduction we have $x = y$ for any $x \equiv y$ and for any $(x, y) \in R$ in particular. □

Any bisimulation is also a bisimulation-up-to, so this generalizes Theorem 2.2 in the case of languages. Consequently, the converse of the above principle holds as well. We proceed with a number of proofs based on bisimulation-up-to.

Example 3.4. Recall the relation $R = \{(x^*x + 1, x^*) \mid x \in \mathcal{P}(A^*)\}$ from the beginning of this section. As we have seen, the a-derivatives are $x_a(x^*x + 1)$

and $x_a x^*$, which are not related by R; however they *are* related by $\equiv_R$. So R is a bisimulation-up-to, and consequently $x^*x + 1 = x^*$. Moreover, the relation $\{(xx^* + 1, x^*) \mid x \in \mathcal{P}(A^*)\}$ from the beginning of this section is a bisimulation-up-to as well; there, the derivatives are equal and thus related by $\equiv_R$.

Thus we have covered coinductively the soundness of the star unfolding axioms (10) and (11).

Example 3.5. For the axiom $z + yx \subseteq x \to y^*z \subseteq x$ consider

$$R = \{(y^*z + x, x) \mid z + yx \subseteq x;\ x, y, z \in \mathcal{P}(A^*)\}\,.$$

Let x, y, z be such languages; notice that $z+yx+x = x$. Since $o(z+yx+x) = o(x)$ it follows that $o(z + x) = o(x)$ so $o(y^*z + x) = o(x)$. For any alphabet letter a we have

$$\begin{aligned}(y^*z + x)_a &= y_a y^* z + z_a + x_a = y_a y^* z + z_a + (z + yx + x)_a \\ &= y_a y^* z + z_a + z_a + y_a x + o(y)x_a + x_a \\ &= y_a(y^*z + x) + z_a + o(y)x_a + x_a \\ &\equiv_R y_a x + z_a + o(y)x_a + x_a \\ &= (z + yx + x)_a = x_a\end{aligned}$$

So R is a bisimulation-up-to, proving $z + yx \subseteq x \to y^*z + x = x$.

In fact, the above way of dealing with language inclusion by reducing it to equality is, in general, not the most efficient one. Indeed, in Section 5 we will introduce *simulation-up-to* which allows to deal with inequations more directly, and reprove the above example in a shorter way.

Example 3.6. Arden's rule, in a special case,[4] states that if $x = yx+z$ for some languages x, y and z, and y does not contain the empty word, then $x = y^*z$. In order to prove its validity coinductively, let x, y, z be languages such that $\varepsilon \notin y$ and $x = yx + z$, and let $R = \{(x, y^*z)\}$. Then $o(y) = 0$, so $o(x) = o(yx + z) = (o(y) \wedge o(x)) \vee o(z) = (0 \wedge o(x)) \vee o(z) = o(z) = 1 \wedge o(z) = o(y^*) \wedge o(z) = o(y^*z)$ and for any $a \in A$:

$$x_a = (yx + z)_a = y_a x + o(y) \cdot x_a + z_a = y_a x + z_a \equiv_R y_a y^* z + z_a = (y^*z)_a\,.$$

So R is a bisimulation-up-to, proving Arden's rule.

While Arden's rule is not extremely difficult to prove without coinduction either, the textbook proofs are significantly longer and arguably more involved than the above proof, which is not much more than taking derivatives combined with a tiny bit of algebraic reasoning. Nevertheless this coinductive proof is completely precise. Giving a formal proof without our methods is not trivial at all; see, e.g., [10] for the discussion of a proof within the theorem prover Isabelle.

[4]We consider a more general version of Arden's rule in Section 5.

In fact, [30] already contains a coinductive proof of Arden's rule; however, this is based on a bisimulation (in contrast to our proof which is based on a bisimulation-up-to). Indeed, in [30] the infinite relation $\{(ux+v, uy^*z+v) \mid u, v \in \mathcal{P}(A^*)\}$ is used, requiring more work in checking the bisimulation conditions. In that case one essentially closes the relation under (certain) contexts manually; the coinduction-up-to principle does this in a general and systematic fashion.

Example 3.7. Let us prove that for any language x, we have $xx = 1 \to x = 1$. Assume $xx = 1$ and let $R = \{(x, 1)\}$. Since $o(xx) = o(1) = 1$ also $o(x) = 1 = o(1)$. We show that the derivatives of x and 1 are equal, turning R into a bisimulation-up-to. For any $a \in A$: $x_a x + x_a = x_a x + o(x)x_a = (xx)_a = 1_a = 0$. One easily proves that this implies $x_a = 0$ (for example by showing that $\{(y, 0) \mid z, y \in \mathcal{P}(A^*), z + y = 0\}$ is a bisimulation). Thus $x_a = 0 = 1_a$, so $x_a \equiv_R 1_a$.

Example 3.8. We prove the soundness of the axiom $xx = x \to x^* = 1 + x$, by establishing a bisimulation-up-to. This axiom was used by Boffa in his complete axiomatization of equivalence of regular expressions [1] (see also [11] for a clear exposition). Let x be a language for which $xx = x$ and consider the relation $R = \{(x^*, 1+x)\}$. Indeed, $o(x^*) = 1 = o(1+x)$, and for any $a \in A$: $(x^*)_a = x_a x^* \equiv_R x_a(1+x) = x_a + x_a x = x_a + o(x)x_a + x_a x = x_a + (xx)_a = x_a + x_a = x_a$.

In fact the above axiom can also easily be proved by induction. In practice, one wants to combine inductive and coinductive methods.

4. Bisimulation-up-to in general

We proceed to generalize the results of the previous section, from regular operations to a large class of operations: those which can be defined by so-called *behavioural differential equations*. Then in Section 4.1 and Section 4.2 below we consider examples involving complement and intersection, and shuffle (closure) respectively.

A *signature* Σ is a collection of operator names $\hat{\sigma} \in \Sigma$ with associated arities[5] $|\hat{\sigma}| \in \mathbb{N}$. One can associate to a signature Σ a collection of functions

$$\{\sigma : \mathcal{P}(A^*)^{|\hat{\sigma}|} \to \mathcal{P}(A^*)\}_{\hat{\sigma} \in \Sigma}\,.$$

In the sequel, every family of functions for a signature will be of the above type (on languages), and so we will simply write $\{\sigma\}_{\hat{\sigma} \in \Sigma}$ for such a family. In order to distinguish between syntax and semantics we will write $\hat{\sigma}$ for function names and σ for actual functions.

We define a general congruence closure with respect to a signature:

Definition 4.1. For a relation $R \subseteq \mathcal{P}(A^*) \times \mathcal{P}(A^*)$, define the *congruence closure* $\equiv_R^\Sigma$ of R w.r.t. a family of functions $\{\sigma\}_{\hat{\sigma} \in \Sigma}$ as the least relation $\equiv$ satisfying the following rules:

$$\frac{x\,R\,y}{x \equiv y} \qquad \frac{}{x \equiv x} \qquad \frac{x \equiv y}{y \equiv x} \qquad \frac{x \equiv y \qquad y \equiv z}{x \equiv z}$$

$$\frac{x_1 \equiv y_1 \qquad \dots \qquad x_n \equiv y_n}{\sigma(x_1, \dots, x_n) \equiv \sigma(y_1, \dots, y_n)} \quad \text{for each } \hat{\sigma} \in \Sigma, n = |\hat{\sigma}|$$

The congruence closure for the regular operators (Definition 3.1) is a special case of the above definition. Given the above congruence closure, we define bisimulation-up-to with respect to a given signature, generalizing Definition 3.2:

Definition 4.2. A relation $R \subseteq \mathcal{P}(A^*) \times \mathcal{P}(A^*)$ is a *bisimulation-up-to (w.r.t. $\{\sigma\}_{\hat{\sigma} \in \Sigma}$)*, if for any $(x, y) \in R$:

1. $o(x) = o(y)$, and
2. for all $a \in A$: $x_a \equiv_R^\Sigma y_a$.

where $\equiv_R^\Sigma$ is the congruence closure w.r.t. $\{\sigma\}_{\hat{\sigma} \in \Sigma}$.

Unfortunately, while the coinduction-up-to principle in Theorem 3.3 shows us that bisimulation-up-to is a sound proof technique in the case of the regular operations, in general, for arbitrary operations, this is not the case.

Example 4.3. We define a function h on languages that is not sound for bisimulation-up-to. This is a simple adaptation of a unary operator on processes, introduced in [27] for a similar purpose. Assume for simplicity a singleton alphabet $\{a\}$. Consider the unary function h on languages, defined as follows:

$$h(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases}$$

Now notice that $0_a = 0 = h(0)$ and $a_a = 1 = h(a)$ (and $o(0) = 0 = o(a)$). Consequently the relation $R = \{(0, a)\}$ is a bisimulation-up-to, while $0 \neq a$, so bisimulation-up-to with respect to h is not sound.

[5]For notational convenience we assume that all operations have finite arity, but all the results hold for non-finitary operations – such as the infinite sum – as well.

We will now introduce a general condition on the operations involved under which we have a valid associated coinduction-up-to principle. In order to proceed we define the set of *terms* over a signature Σ and a set of variables V as least set $T_\Sigma V$ such that

- $V \subseteq T_\Sigma V$, and
- $\hat{\sigma}(t_1, \ldots, t_n) \in T_\Sigma V$ for any $\hat{\sigma}$ in Σ (of arity n), and any $t_1, \ldots, t_n \in T_\Sigma V$.

Given a family of functions $\{\sigma\}_{\hat{\sigma} \in \Sigma}$ we define an interpretation

$$I : T_\Sigma(\mathcal{P}(A^*)) \to \mathcal{P}(A^*)$$

by induction: $I(L) = L$ and $I(\hat{\sigma}(t_1, \ldots, t_n)) = \sigma(I(t_1), \ldots, I(t_n))$. We will use the standard definition of substitution in t of a variable x for a term u, denoted $t[x := u]$. For sequences $x_1, \ldots, x_n$ and $u_1, \ldots, u_n$ we abbreviate multiple substitution $t[x_1 := u_1, \ldots, x_n := u_n]$ by $t[x_i := u_i]$.

Our soundness condition depends on characterizing operations in terms of behavioural differential equations [32]. Informally this means that one specifies the output of an operation in terms of the outputs of the arguments, and the derivatives as an expression involving the arguments, their derivatives and their outputs. The equations in Lemma 2.3 form a concrete example. We note that this format is essentially the stream GSOS format of [14, 17], adapted to deterministic automata. A small variation is that output values occur in the derivatives rather than as a condition for the derivative.

Definition 4.4. We say a family of functions $\{\sigma\}_{\hat{\sigma} \in \Sigma}$ can be given by *behavioural differential equations* if for each function f of arity n there are functions

$$\begin{aligned} &i : 2^n \to 2 \\ &d : A \to T_\Sigma(\bar{u}, \bar{u} \times A, \bar{o}) \end{aligned}$$

where 2^n denotes the set of n-tuples with binary entries; $\bar{u} = \{u_1, \ldots, u_n\}$ and $\bar{o} = \{o_1, \ldots, o_n\}$ are disjoint collections of n variables, such that for all languages $x_1, \ldots, x_n$:

$$\begin{aligned} o(\sigma(x_1, \ldots, x_n)) &= i(o(x_1), \ldots, o(x_n)) \\ \forall a \in A : \sigma(x_1, \ldots, x_n)_a &= I(d(a)[u_j := x_j, ((u_j, b) := (x_j)_b)_{b \in A}, o_j := o(x_j)]) \,. \end{aligned}$$

The function i specifies the output given the output of the operations, whereas the function d specifies, for each alphabet letter, the derivative. This derivative is given as a term; intuitively a variable u_i represents the i-th argument of the operation, a variable (u_i, a) represents the a-derivative of the i-th argument, and a variable o_i represents its output. Indeed Lemma 2.3 witnesses that the regular operations can be given by behavioural differential equations, since it characterizes the operations precisely in this way. So the following theorem generalizes the coinduction-up-to principle of Theorem 3.3:

Theorem 4.5 (Coinduction-up-to)**.** *If $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ can be given by behavioural differential equations, then for any relation R which is a bisimulation-up-to w.r.t. $\{\sigma\}_{\hat{\sigma}\in\Sigma}$: if $(x, y) \in R$ then $x = y$.*

Proof. Similarly to the proof of Theorem 3.3 we show that the congruence closure $\equiv$ of R is a bisimulation, by proving by structural induction that (1) $o(x) = o(y)$ and (2) $x_a = y_a$ holds for any $(x, y) \in \equiv$. The base cases, i.e., if $x = y$ or $(x, y) \in R$, are the same as in Theorem 3.3.

Let $\hat{\sigma} \in \Sigma$, $n = |\hat{\sigma}|$, let i and d be the functions from Definition 4.4 associated to σ which exist since $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ can be given by behavioural differential equations, and suppose we have languages $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ such that for all j: $x_j \equiv y_j$, $o(x_j) = o(y_j)$ and for all $a \in A$: $(x_j)_a \equiv (y_j)_a$. Then

$$o(\sigma(x_1, \ldots, x_n)) = i(o(x_1), \ldots, o(x_n)) = i(o(y_1), \ldots, o(y_n)) = o(\sigma(y_1, \ldots, y_n))\,.$$

For any $a \in A$:

$$\begin{aligned}\sigma(x_1, \ldots, x_n)_a &= I(d(a)[u_j := x_j, ((u_j, b) := (x_j)_b)_{b\in A}, o_j := o(x_j)])\\ &\equiv I(d(a)[u_j := y_j, ((u_j, b) := (y_j)_b)_{b\in A}, o_j := o(y_j)])\\ &= \sigma(y_1, \ldots, y_n)_a\end{aligned}$$

where the terms are related by $\equiv$ since for all j: $x_j \equiv y_j$, $(x_j)_b \equiv (y_j)_b$ for all $b \in A$, and $o(x_j) \equiv o(y_j)$ (the latter holds since $o(x_j) = o(y_j)$). The symmetry and transitivity rules are again easy to treat. This concludes the proof that $\equiv$ is a bisimulation, and the desired result follows by coinduction. □

Bisimulation-up-to with respect to the function h, introduced in Example 4.3, is not sound, as we have seen. Indeed h cannot be given by behavioural differential equations, since the output $o(h(x))$ depends on the entire language x an not only on its output.

In the following, we will recall behavioural differential equations for language complement and intersection (Section 4.1), and shuffle (closure) (Section 4.2), and apply Theorem 4.5 to give a number of example proofs based on bisimulation-up-to.

4.1. Language equations with complement and intersection

Context-free languages can be expressed in terms of certain types of language equations [12]. For example, the language $\{a^n b^n \mid n \in \mathbb{N}\}$ is the unique language x such that $x = axb + 1$. Our coinductive techniques can directly be applied to languages defined in such a way, and so we are able to reason about (equivalence of) context-free languages in a novel manner.

Example 4.6. Let x, y, z be languages such that $x = ayzb + 1$, $y = azxb + 1$ and $z = axyb + 1$. Without thinking of what possible concrete descriptions of x, y and z can be, let us show, by coinduction, that $x = y = z$. We use the relation $R = \{(x, y), (y, z)\}$. Obviously $o(x) = o(y)$ and $o(y) = o(z)$. Moreover for any alphabet letter b other than a, we have $x_b = 0 = y_b$ and $y_b = 0 = z_b$. For the a-derivatives we have $x_a = yzb \equiv_R zyb \equiv_R zxb = y_a$ and similarly for (y, z); so R is a bisimulation-up-to, proving that $x = y = z$.

We proceed to incorporate complement and intersection, defined as $\overline{x} = \{w \mid w \notin x\}$ and $x \wedge y = \{w \mid w \in x \text{ and } w \in y\}$ respectively. Language equations including these additional operators can be used to give semantics to *conjunctive*- and *Boolean grammars* [25]. Complement and intersection have a known characterization in terms of outputs and derivatives as well [5]:

Lemma 4.7. *For any two languages x, y and for any $a \in A$:*

$$\begin{array}{ll} o(\overline{x}) = \neg o(x) & \qquad \overline{x}_a = \overline{x_a} \\ o(x \wedge y) = o(x) \wedge o(y) & \qquad (x \wedge y)_a = x_a \wedge y_a \end{array}$$

As a consequence, we have bisimulation and coinduction to our disposal to show equivalence of languages defined in terms of systems of equations involving these additional operators. The above characterization, in fact, is in terms of behavioural differential equations; and as such, we immediately obtain from Theorem 4.5 the soundness of bisimulation-up-to.

We have already seen that $\langle \mathcal{P}(A^*), 0, 1, +, \cdot, * \rangle$ is a Kleene algebra; it is useful to know that $\langle \mathcal{P}(A^*), 0, A^*, \overline{(-)}, +, \wedge \rangle$ is a Boolean algebra. Moreover, below we will need the following property, which holds for any language x and $a \in A$:

$$\overline{xa} = \overline{x}a + \sum_{b \in A \setminus \{a\}} A^* b + 1 \tag{14}$$

Example 4.8. There are unique languages x and y such that

$$x = axa + bxb + a + b + 1 \qquad\qquad y = aya + byb + aA^*b + bA^*a$$

x is the language of *palindromes*, i.e., words which are equal to their own reverse. We claim that y must be the language of all *non*-palindromes, i.e., $y = \overline{x}$. We proceed to prove this formally by showing that the relation $R = \{(\overline{x}, y)\}$ is a bisimulation-up-to. The outputs are easily seen to be equal: $o(\overline{x}) = \neg o(x) = \neg o(1) = 0 = o(y)$. We consider the a-derivatives; the b-derivatives are of course similar. In the fourth step we use (14).

$$\begin{aligned} \overline{x}_a &= \overline{x_a} = \overline{xa + 1} = \overline{xa} \wedge \overline{1} = (\overline{x}a + A^*b + 1) \wedge \overline{1} \\ &\equiv_R (ya + A^*b + 1) \wedge \overline{1} = ya \wedge \overline{1} + A^*b \wedge \overline{1} + 1 \wedge \overline{1} = ya + A^*b = y_a \end{aligned}$$

So R is a bisimulation-up-to, proving that y indeed is the complement of x.

4.2. Shuffle (closure)

The *shuffle* operation is defined on words w, v inductively as follows: $w \odot \varepsilon = \varepsilon \odot w = w$ and $aw \odot bv = a(w \odot bv) + b(aw \odot v)$ for any alphabet letters a, b. This is extended to languages x, y as $x \odot y = \sum_{w \in x, v \in y} w \odot v$. The *shuffle closure* is defined as

$$x^{\circledast} = \sum_{i=0}^{\infty} x^{\odot i}$$

where $x^{\odot_i}$ is given inductively as $x^{\odot_0} = 1$ and $x^{\odot_{i+1}} = x \odot x^{\odot_i}$. Notice that the shuffle closure is in fact very similar to the Kleene star; the difference is that here shuffle is used instead of concatenation. Both shuffle and shuffle closure can be characterized in terms of behavioural differential equations, as stated by the following lemma. We include a proof for the sake of completeness.

Lemma 4.9. *For any two languages x, y and for any $a, b \in A$:*

$$\begin{array}{ll} (x \odot y)_a = x_a \odot y + x \odot y_a & o(x \odot y) = o(x) \wedge o(y) \\ (x^{\circledast})_a = x_a \odot x^{\circledast} & o(x^{\circledast}) = 1 \end{array}$$

Proof. We will only consider the derivatives, and first treat $(x \odot y)_a = x_a \odot y + x \odot y_a$. Notice that $(w \odot v)_a = w_a \odot v + w \odot v_a$ holds for any two words w, v, which one can prove formally by induction on the length of w and v. Now for any languages x, y we have

$$\begin{aligned} (x \odot y)_a &= (\textstyle\sum_{w \in x, v \in y} w \odot v)_a \\ &= \textstyle\sum_{w \in x, v \in y} (w \odot v)_a \\ &= \textstyle\sum_{w \in x, v \in y} (w_a \odot v + w \odot v_a) \\ &= \textstyle\sum_{w \in x, v \in y} (w_a \odot v) + \sum_{w \in x, v \in y} (w \odot v_a) \\ &= \textstyle\sum_{w \in x_a, v \in y} (w \odot v) + \sum_{w \in x, v \in y_a} (w \odot v) \\ &= x_a \odot y + x \odot y_a \end{aligned}$$

For $(x^{\circledast})_a = x_a$, we will use that shuffle distributes over infinite sum (union), which is easy to establish. Moreover we will use that $(x^{\odot_{i+1}})_a = x_a \odot x_i^{\odot}$, which can be shown by induction. Now

$$\begin{aligned} (x^{\circledast})_a &= (\textstyle\sum_{i=0}^{\infty} x^{\odot_i})_a \\ &= \textstyle\sum_{i=0}^{\infty} ((x^{\odot_i})_a) \\ &= \textstyle\sum_{i=0}^{\infty} (x_a \odot x^{\odot_i}) \\ &= x_a \odot \textstyle\sum_{i=0} x^{\odot_i} \\ &= x_a \odot x^{\circledast} \end{aligned}$$

□

Notice again the similarity between the above behavioural differential equations and those for concatenation and Kleene star. We proceed to exhibit bisimulation-up-to techniques for the shuffle (closure).

We recalled in Section 2 that the set of all languages together with the operations of sum, concatenation, Kleene star and the constants 1 and 0 forms a Kleene algebra. In fact, by replacing concatenation and Kleene star by shuffle and shuffle closure respectively, one obtains a *commutative* Kleene algebra, meaning that all the KA axioms are satisfied and additionally the shuffle is commutative.

Example 4.10. Let x be any language; we will show that $x^{\circledast} = x \odot x^{\circledast} + 1$. To this end let $R = \{(x^{\circledast}, x \odot x^{\circledast} + 1\}$. Then $o(x^{\circledast}) = 1 = o(x \odot x^{\circledast} + 1)$. Moreover for any alphabet letter a:

$$\begin{array}{rcll}
(x^{\circledast})_a & = & x_a \odot x^{\circledast} & \text{Lemma 4.9} \\
 & = & x_a \odot (x^{\circledast} + x^{\circledast}) & \text{idempotence} \\
 & \equiv_R & x_a \odot (x^{\circledast} + x \odot x^{\circledast} + 1) & \\
 & = & x_a \odot (x^{\circledast} + x \odot x^{\circledast}) & x^{\circledast} + 1 = x^{\circledast} \\
 & = & x_a \odot x^{\circledast} + x_a \odot x \odot x^{\circledast} & \text{distributivity} \\
 & = & x_a \odot x^{\circledast} + x \odot x_a \odot x^{\circledast} & \text{commutativity} \\
 & = & (x \odot x^{\circledast} + 1)_a & \text{Lemmas 4.9, 2.3}
\end{array}$$

so R is a bisimulation-up-to, proving $x^{\circledast} = x \odot x^{\circledast} + 1$.

5. Simulation(-up-to)

So far we have focused on techniques for showing equality of languages. Of course, one can also apply these methods to prove language inclusion, since $x \subseteq y$ iff $x + y = y$. However, there is a more direct way: instead of bisimulations, one can establish *simulations*, which in practice turns out to be easier for proving inequalities. In this section we first recall this notion, and then introduce up-to techniques for simulation.

Definition 5.1. Let (S, o, δ) be a deterministic automaton. A *simulation* is a relation $R \subseteq S \times S$ such that for any $(s, t) \in R$:

1. $o(s) \leq o(t)$, and
2. for all $a \in A$: $(\delta(s, a), \delta(t, a)) \in R$.

Notice that the only difference with bisimulation is that the first condition is relaxed: if s is a final state then t should be final as well, but if s is not final then the output of t does not matter.

Theorem 5.2 (Coinduction (for simulation))**.** *If R is a simulation then for any pair of states $(s, t) \in R : l(s) \subseteq l(t)$.*

Recall from Section 2 that the set of all languages forms a deterministic automaton. By the above principle we have, for any two languages x and y, that $x \subseteq y$ whenever $(x, y) \in R$ for a simulation R on this automaton. Thus simulation is a concrete proof principle for language inclusion, just like bisimulation is a proof principle for language equality.

We proceed directly to introduce up-to techniques for simulation. In order to do so we define the *precongruence closure* of a relation R on languages with respect to a signature. This is similar to the congruence closure of Definition 4.1; the difference is that this closure is not symmetric, and it relates x to y whenever x is included in y.

Definition 5.3. For a relation $R \subseteq \mathcal{P}(A^*) \times \mathcal{P}(A^*)$, define the *precongruence closure* $\leqq_R^\Sigma$ of R w.r.t. a family of functions $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ as the least relation $\leqq$ satisfying the following rules:

$$\frac{x\,R\,y}{x \leqq y} \qquad \frac{x \subseteq y}{x \leqq y} \qquad \frac{x \leqq y \qquad y \leqq z}{x \leqq z}$$

$$\frac{x_1 \leqq y_1 \qquad \dots \qquad x_n \leqq y_n}{\sigma(x_1, \dots, x_n) \leqq \sigma(y_1, \dots, y_n)} \qquad \text{for each } \hat{\sigma} \in \Sigma, n = |\hat{\sigma}|$$

If Σ is clear from the context we write $\leqq_R$ for $\leqq_R^\Sigma$.

The notion of simulation-up-to is as expected:

Definition 5.4. A relation $R \subseteq \mathcal{P}(A^*) \times \mathcal{P}(A^*)$ is a *simulation-up-to* (w.r.t. $\{\sigma\}_{\hat{\sigma}\in\Sigma}$), if for any $(x, y) \in R$:

1. $o(x) \leq o(y)$, and
2. for all $a \in A$: $x_a \leqq_R^\Sigma y_a$.

where $\leqq_R^\Sigma$ is the precongruence closure w.r.t. $\{\sigma\}_{\hat{\sigma}\in\Sigma}$.

The soundness criterion for bisimulation-up-to, namely that the operations can be given by behavioural differential equations, turns out not to be strong enough for simulation-up-to, as witnessed by the following example.

Example 5.5. We have seen that the complement operation can be easily given by behavioural differential equations. Consider the relation $R = \{(\overline{aA^*}, \overline{0})\}$. We have $o(\overline{aA^*}) = 0 = o(\overline{0})$. Moreover $(\overline{aA^*})_a = \overline{A^*} = \overline{0}$ and $\overline{0}_a = \overline{0} = \overline{A^*}$. Since $0 \leq A^*$, we have $\overline{0} \leqq_R \overline{A^*}$ and thus $(\overline{aA^*})_a \leqq_R \overline{0}_a$, showing that R is a simulation-up-to. But clearly $\overline{aA^*} \nsubseteq \overline{0}$, so simulation-up-to with respect to language complement is not a sound proof principle.

The solution is to additionally require the operations under consideration to satisfy a monotonicity condition.

Definition 5.6. A family of operations $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ can be given by *monotone behavioural differential equations* if

1. $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ can be given by behavioural differential equations, and
2. for each $\sigma \in \Sigma$: the associated (output) function $i : 2^n \to 2$ is monotone, i.e., if $b_j \leq b'_j$ for all j with $1 \leq j \leq n$ then $i(b_1, \dots, b_n) \leq i(b'_1, \dots, b'_n)$.

Behavioural differential equations are essentially stream GSOS [14, 17]; the monotonicity condition rules out negative premises with respect to output values, and as such is reminiscent of positive GSOS formats.

Theorem 5.7 (Coinduction-up-to (for simulation))**.** *If $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ can be given by monotone behavioural differential equations, then for any relation R which is a simulation-up-to w.r.t. $\{\sigma\}_{\hat{\sigma}\in\Sigma}$: if $(x, y) \in R$ then $x \subseteq y$.*

Proof. The proof is mostly similar to that of Theorem 4.5: one proves by induction that $\leqq$, the precongruence closure of R, is a simulation. The only difference is the first part of the inductive step, which concerns the output. Suppose σ is an operation with arity n, from a family $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ of operations given by monotone behavioural differential equations, and let i be its output function. Let $x_1, \ldots, x_n$ and $y_1, \ldots y_n$ be languages such that for all j: $o(x_j) \leq o(y_j)$. Then

$$o(\sigma(x_1, \ldots, x_n)) = i(o(x_1), \ldots, o(x_n)) \leq i(o(y_1), \ldots, o(y_n)) = o(\sigma(y_1, \ldots, y_n))$$

where we use the assumption that i is monotone. □

Example 5.8. The general version of Arden's rule states that given languages y and z, the *least* solution of $x = yx + z$ is y^*z. If $\varepsilon \notin y$ then it is the unique one – as we have seen in Example 3.6. For the proof, first notice that y^*z is indeed a solution since $y^*z = (yy^*+1)z = yy^*z+z$. In order to show it is the least one, let x be any language such that $x = yx+z$ and consider the relation $R = \{(y^*z, x)\}$. Then R is a simulation-up-to, since $o(y^*z) = o(z) \leq o(yx + z) = o(x)$ and for any alphabet letter a:

$$(y^*z)_a = y_a y^* z + z_a \leqq_R y_a x + z_a \subseteq y_a x + o(y)x_a + z_a = (yx + z)_a = x_a \,.$$

Thus y^*z is the least solution. Finally suppose $\varepsilon \notin y$, and u and v are both solutions; then $\{(u, v)\}$ is easily shown to be a bisimulation-up-to.

The reader is invited to formulate and prove a version of Arden's rule, where shuffle and shuffle closure (Section 4.2) replace concatenation and Kleene star. We proceed with an axiom used in concurrency theory [16], which concerns the interplay between shuffle and concatenation:

Example 5.9. The *exchange law* connects shuffle and concatenation as follows:

$$(w \odot x)(y \odot z) \subseteq (wy) \odot (xz)$$

for any languages w, x, y, z. Consider the relation

$$R = \{((w \odot x)(y \odot z), (wy) \odot (xz)) \mid w, y, x, z \in \mathcal{P}(A^*)\} \,.$$

Then

$$o((w \odot x)(y \odot z)) = o(w) \wedge o(x) \wedge o(y) \wedge o(z) = o((wy) \odot o(xz))$$

and for any alphabet letter a:

$$\begin{aligned}
&((w \odot x)(y \odot z))_a \\
&= (w_a \odot x + w \odot x_a)(y \odot z) + o(w \odot x)(y_a \odot z + y \odot z_a) \\
&= (w_a \odot x)(y \odot z) + (w \odot x_a)(y \odot z) + (o(w) \odot o(x))(y_a \odot z) \\
&\quad + (o(w) \odot o(x))(y \odot z_a) \\
&\leqq_R (w_a y) \odot (xz) + (wy) \odot (x_a z) + (o(w)y_a) \odot (o(x)z) + (o(w)y) \odot (o(x)z_a) \\
&\subseteq (w_a y) \odot (xz) + (wy) \odot (x_a z) + (o(w)y_a) \odot (xz) + (wy) \odot (o(x)z_a) \\
&= (w_a y + o(w)y_a) \odot (xz) + (wy) \odot (x_a z + o(x)z_a) \\
&= ((wy) \odot (xz))_a
\end{aligned}$$

which shows that R is a simulation-up-to and proves the exchange law.

The shuffle and concatenation operators on languages are essentially analogous to sequential and parallel composition in process algebra. In fact, in the context of strong similarity of processes the exchange law holds as well.

The proof in the above example is clearly easier than one where the inclusion would be reduced to checking equality by means of bisimilarity. In order to further compare bisimulation and simulation, we revisit Example 3.5:

Example 5.10. Recall the axiom $z + yx \subseteq x \to y^*z \subseteq x$; we will now prove it by showing that $R = \{(y^*z, x) \mid z + yx \subseteq x;\ x, y, z \in \mathcal{P}(A^*)\}$ is a simulation. Let x, y, z be such languages; then $o(z) \leq o(z + yx) \leq o(x)$ so $o(y^*z) \leq o(x)$. For any alphabet letter a:

$$(y^*z)_a = y_a y^* z + z_a \leqq_R y_a x + z_a \subseteq y_a x + o(y)x_a + z_a = (z + yx)_a \subseteq x_a$$

indeed turning R into a simulation-up-to, and proving the result in a clearly more efficient way than in Example 3.5.

One might expect that the axiom $z + xy \subseteq x \to zy^* \subseteq x$ is similar, but due to the asymmetry of the derivative of concatenation it is not. We present a proof below by simulation-up-to.

Example 5.11. In order to prove $z + xy \subseteq x \to zy^* \subseteq x$ consider the relation $R = \{(zy^*, x) \mid z + xy \subseteq x;\ x, y, z \in \mathcal{P}(A^*)\}$. Let x, y, z be such languages; then $o(z) \leq o(x)$, so $o(zy^*) \leq o(x)$. For any $a \in A$, we have

$$(zy^*)_a = z_a y^* + o(z) y_a y^* = (z_a + o(z) y_a) y^*$$

Now in order to see that this is related by $\leqq_R$ to x_a, we start with our assumption $z + xy \subseteq x$ and compute derivatives: $(z + xy)_a \subseteq x_a$, so $z_a + x_a y + o(x) y_a \subseteq x_a$. Reformulating this as $(z_a + o(x)y_a) + x_a y \subseteq x_a$, we have

$$((z_a + o(x)y_a)y^*, x_a) \in R\,.$$

Since $o(z) \leq o(x)$ we thus obtain

$$(zy^*)_a = (z_a + o(z)y_a)y^* \subseteq (z_a + o(x)y_a)y^* \leqq_R x_a$$

as desired, showing that R is a simulation-up-to and proving the axiom.

The above proof makes essential use of the fact that $o(z) \leq o(x)$, and as such it seems non-trivial to come up with a similar (short) proof based on bisimulation-up-to.

6. Behavioural differential equations and causal functions

We have established behavioural differential equations as a format providing a sufficient condition for soundness of bisimulation-up-to techniques. Our format is an extension of the similar one for streams, as given in [14, 20]. There, it is shown that functions adhering to this format are *causal*, and vice versa. For operators on streams, this informally means that the n-th value of the output stream depends only on the first n values of the arguments. As such, this notion has a straightforward generalisation to languages. In the present section we adapt the above result to operations on languages, obtaining causality of functions as an equivalent, *semantic* condition for soundness of up-to techniques. This also provides a connection to [9], where up-to techniques for streams are considered; there, causality is taken as a sufficient condition for soundness.

In this section we assume a finite alphabet A to ease the notation. For any language x and any $k \in \mathbb{N}$ we will write

$$x|_k = \{w \in x \mid |w| \leq k\}$$

and we define the relation $\approx_k$ between languages as follows:

$$x \approx_k y \text{ iff } x|_k = y|_k \,.$$

A function $\sigma : \mathcal{P}(A^*)^n \to \mathcal{P}(A^*)$ is *causal* if for any languages $x_1, \ldots, x_n$, $y_1, \ldots, y_n$ and for any $k \in \mathbb{N}$:

$$x_1 \approx_k y_1, \ldots, x_n \approx_k y_n \text{ implies } \sigma(x_1, \ldots, x_n) \approx_k \sigma(y_1, \ldots, y_n) \,.$$

In order to simplify the presentation we will use an equivalent characterization: σ is causal iff for any languages $x_1, \ldots, x_n$ and $k \in \mathbb{N}$:

$$\sigma(x_1, \ldots, x_n) \approx_k \sigma(x_1|_k, \ldots, x_n|_k) \,.$$

Lemma 6.1. *The family of all causal functions can be given by behavioural differential equations.*

Proof. The core of the proof is that the derivatives of causal functions can be expressed in terms of causal functions again. We only show how this works for a unary function $\sigma : \mathcal{P}(A^*) \to \mathcal{P}(A^*)$; the extension to other arities is straightforward. Let $A = \{a_1, \ldots, a_l\}$ be a finite alphabet. Consider, for an alphabet letter $a \in A$, the function

$$\tilde{\sigma}_a : \mathcal{P}(A^*)^{l+1} \to \mathcal{P}(A^*)$$

defined as $\tilde{\sigma}_a(z, y_1, \ldots, y_l) = \sigma(o(z) + a_1 y_1 + \ldots + a_l y_l)_a$. Then $\tilde{\sigma}_a$ is causal, and it follows that

$$\sigma(x)_a = \tilde{\sigma}_a(o(x), x_{a_1}, \ldots, x_{a_l}) \,.$$

□

In order to prove the converse, we need the following technical result.

Lemma 6.2. *Let $k \in \mathbb{N}$. Suppose for all σ in some family $\{\sigma\}_{\hat{\sigma} \in \Sigma}$, and for all languages $x_1, \ldots, x_n$ (where $n = |\sigma|$) we have*

$$\sigma(x_1, \ldots, x_n) \approx_k \sigma(x_1|_k, \ldots, x_n|_k) \,.$$

Then for any term $t \in T_\Sigma(u_1, \ldots, u_m)$ over operators in Σ, and any languages $x_1, \ldots, x_m$:

$$I(t[u_i := x_i]) \approx_k I(t[u_i := x_i|_k])\,.$$

Proof. By structural induction on terms. For the base case, if t is a variable then $I(t[u_i := x_i]) = x$ and $I(t[u_i := x_i|_k]) = x|_k$, and clearly $x \approx_k x|_k$. The induction step follows by the assumption. □

We can now state and prove our main result of this section.

Theorem 6.3. *A function $\sigma : \mathcal{P}(A^*)^n \to \mathcal{P}(A^*)$ is causal if and only if it is contained in a family of functions $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ which can be given by behavioural differential equations.*

Proof. From left to right, the result follows from Lemma 6.1. For the other direction, let $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ be given by behavioural differential equations. We prove that for any σ (with $n = |\sigma|$) and any k we have $\sigma(x_1, \ldots, x_n) \approx_k \sigma(x_1|_k, \ldots, x_n|_k)$, by induction on k.

The base case follows from the fact that the output of σ is given in terms of a function i applied to the output of its arguments. Now suppose it holds for some $k \in \mathbb{N}$. Then for any $a \in A$, using Lemma 6.2 one can obtain

$$\sigma(x_1, \ldots, x_n)_a \approx_k \sigma(x_1|_{k+1}, \ldots, x_n|_{k+1})_a$$

because of the characterization of the derivatives (σ is given by BDEs) and the fact that $(x_a)|_k = (x|_{k+1})_a$. Thus, for any word w of length k and any $a \in A$:

$$\begin{aligned} aw \in \sigma(x_1, \ldots, x_n) &\text{ iff } w \in \sigma(x_1, \ldots, x_n)_a \\ &\text{ iff } w \in \sigma(x_1|_{k+1}, \ldots, x_n|_{k+1})_a \\ &\text{ iff } aw \in \sigma(x_1|_{k+1}, \ldots, x_n|_{k+1})\,. \end{aligned}$$

Consequently $\sigma(x_1, \ldots, x_n) \approx_{k+1} \sigma(x_1|_{k+1}, \ldots, x_n|_{k+1})$, which proves the induction step. □

By Theorem 4.5 and the above result we directly obtain causality as a sufficient condition for the soundness of bisimulation-up-to:

Corollary 6.4. *If every function of a family $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ is causal, then bisimulation-up-to w.r.t. $\{\sigma\}_{\hat{\sigma}\in\Sigma}$ is sound.*

7. Discussion and related work

The theory of bisimulation and bisimulation-up-to developed in this paper can be viewed as part of the general theory of *coalgebras*. Coalgebra [31] is a general mathematical theory for the uniform study of state-based systems including labelled transition systems but also stream systems, various kinds of (weighted or probabilistic) automata, etc. Indeed, *deterministic automata*, as presented in Section 2, are also a certain type of coalgebras. *Bisimulation* is the canonical notion of equivalence of coalgebras, which, for labelled transition systems, coincides with the classical notion introduced by Milner and Park [22, 26]. In the case of deterministic automata, the associated instance of bisimulation is precisely the one presented in Section 2. The material of that section, as well as the notion of *simulation* and the corresponding principle of coinduction, is from [30], which contains an extensive investigation of automata and languages as coalgebras.

Bisimulation-up-to classically is a family of enhancements of bisimulation for labelled transition systems [27, 33]; it is a rich theory which drastically improves the bisimulation proof method for, e.g., CCS processes. One interesting result based on bisimulation-up-to is the decidability result of [6], on equivalence of context-free processes. For many more applications of up-to techniques in concurrency theory, we refer to [27]. Note that these notions of bisimulation-up-to are different than those considered in the current paper, since our notion of bisimulation for *automata* is different from the classical one for labelled transition systems: it is deterministic and features output of states. It might be possible to obtain our soundness results by encoding automata into labelled transition systems with predicates, but in order to keep the paper self-contained we have chosen not to do so.

Recently the theory of bisimulation-up-to was generalized from labelled transition systems to a large class of coalgebras [2, 28], yielding enhancements of the bisimulation proof method for many different kinds of state-based systems. In fact, the soundness theorems of bisimulation-up-to can be derived from this general theory. In order to show this formally, one would have to show that behavioural differential equations can be represented as so-called *abstract GSOS specifications*, which are a way of defining operational semantics [34]. Then one obtains by the theory of [2, 28] that bisimulation up to congruence for all of these cases is sound, meaning that any bisimulation-up-to can be extended to a bisimulation. For a more abstract view on *simulation*, we refer to [2], which is a framework for up-to techniques for more general coinductive predicates, including simulation.

While we have introduced techniques which are much more widely applicable (as we have shown) than only to regular languages, we proceed to recall some of the related work on checking equivalence of regular expressions. There is a wide range of different tools and techniques tailored towards doing this; we only recall the ones most relevant to our work. CIRC [21] is a general coinductive theorem prover, which can deal with regular expressions. Recently, various algorithms based on Brzozowski derivatives and bisimulations have been implemented in Isabelle [19] and formalized in type theory, yielding an implementation in Coq [8] (while [8] does not mention bisimulations explicitly, their method is based on constructing a bisimulation). Moreover there is another Coq implementation of regular expression equivalence based on partial derivatives [24]. An efficient algorithm for deciding equivalence in Kleene algebra, based on automata but not on derivatives and bisimulations, was recently implemented in Coq as well [4]. Of course, one can reason about regular expressions in Kleene algebra; this is however a fundamentally different approach than the coinductive techniques of the present paper. In [13] a proof system for equivalence of regular expressions is presented, based on bisimulations but not on bisimulation-up-to. In [15] a general coinductive axiomatization of regular expression containment is given, based on an interpretation of regular expressions as types. The authors of [15] instantiate their axiomatization with the main coinductive rule from [13]. The focus of [15] is on constructive proofs based on parse trees of regular expressions; instead, we base ourselves on bisimulations between languages.

The recent [3] introduces an efficient algorithm for checking equivalence of non-deterministic automata based on bisimulation up to congruence, using the algebraic structure of join-semilattices obtained by determinization. Our approach is different in that we consider algebraic structures for *arbitrary* calculi on languages (given by behavioural differential equations). Moreover, we do not focus on the algorithmic aspect, but consider up-to techniques for infinite state systems, so to prove, e.g., (quasi)-equations over arbitrary languages.

If one works with syntactic *terms*, such as regular expressions, rather than with languages, the notion of *bisimulation up to bisimilarity* becomes relevant. In the corresponding proof method, one can relate derivatives, which are then terms, modulo bisimilarity. Since we work directly with languages, in our case this is not necessary; but for dealing with terms our techniques can easily be combined with up-to-bisimilarity (see [28]). Bisimulation up to bisimilarity (alone, without context and equivalence closure) was originally introduced in [23], and in the context of automata and languages *simulation up to similarity* was introduced in [30].

8. Conclusions

We presented *bisimulation-up-to* as a proof method for language equivalence, and *simulation-up-to* for language inclusion. These techniques are sound enhancements of the coinductive proof technique of (bi)simulation whenever the operations under consideration adhere to the format of behavioural differential equations given in this paper – for simulation-up-to, the operations additionally need to satisfy a simple monotonicity condition. We have exemplified our approach with a wide variety of novel proofs of classical results. Finally we have shown that the operations which allow a characterization in terms of behavioural differential equations are precisely the *causal* functions, giving a semantic characterization of functions for which the presented up-to techniques are sound.

The presented proof techniques are very general, and apply to undecidable problems such as language equivalence of context-free grammars. Indeed, automation is not the aim of the present paper. Nevertheless, the present techniques can be seen as a foundation for novel interactive theorem provers, and extensions of fully automated tools such as [8, 19, 21].

References

[1] M. Boffa. Une condition impliquant toutes les identités rationnelles. *ITA*, 29(6):515–518, 1995.

[2] F. Bonchi, D. Petrisan, D. Pous, and J. Rot. Coinduction up-to in a fibrational setting. In Thomas A. Henzinger and Dale Miller, editors, *Proc. CSL-LICS 2014*, page 20. ACM, 2014.

[3] F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In Roberto Giacobazzi and Radhia Cousot, editors, *Proc. POPL 2013*, pages 457–468. ACM, 2013.

[4] T. Braibant and D. Pous. Deciding Kleene algebras in Coq. *Logical Methods in Computer Science*, 8(1), 2012.

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

[6] S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. In R. Cleaveland, editor, *CONCUR*, volume 630 of *LNCS*, pages 138–147. Springer, 1992.

[7] J.H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, 1971.

[8] T. Coquand and V. Siles. A decision procedure for regular expression equivalence in type theory. In J-P. Jouannaud and Z. Shao, editors, *CPP*, volume 7086 of *LNCS*, pages 119–134. Springer, Proc. CPP 2011.

[9] J. Endrullis, D. Hendriks, and M. Bodin. Circular coinduction in coq using bisimulation-up-to techniques. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *Proc. ITP 2013*, volume 7998 of *LNCS*, pages 354–369. Springer, 2013.

[10] S. Foster and G. Struth. Automated analysis of regular algebra. In B. Gramlich, D. Miller, and U. Sattler, editors, *Proc. IJCAR 2012*, volume 7364 of *LNCS*, pages 271–285. Springer, 2012.

[11] S. Foster and G. Struth. On the fine-structure of regular algebra. *Journal of Automated Reasoning*, pages 1–33, 2014.

[12] S. Ginsburg and H.G. Rice. Two families of languages related to ALGOL. *Journal of the ACM*, 9(3):350–371, 1962.

[13] C. Grabmayer. Using proofs by coinduction to find "traditional" proofs. In J.L. Fiadeiro, N. Harman, M. Roggenbach, and J.J.M.M. Rutten, editors, *Proc. CALCO 2005*, volume 3629 of *LNCS*, pages 175–193. Springer, 2005.

[14] H.H. Hansen, C. Kupke, and J.J.M.M. Rutten. Stream differential equations: Specification formats and solution methods. Technical Report No. FM-1404, CWI, 2014.

[15] F. Henglein and L. Nielsen. Regular expression containment: coinductive axiomatization and computational interpretation. In T. Ball and M. Sagiv, editors, *Proc. POPL 2011*, pages 385–398. ACM, 2011.

[16] T. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra and its foundations. *The Journal of Logic and Algebraic Programming*, 80(6):266–296, 2011.

[17] B. Klin. Bialgebras for structural operational semantics: An introduction. *TCS*, 412(38):5043–5069, 2011.

[18] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *Proc. LICS 1991*, pages 214–225. IEEE Computer Society, 1991.

[19] A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *Journal of Automated Reasoning*, 49:95–106, 2012. published online March 2011.

[20] C. Kupke, M. Niqui, and J.J.M.M. Rutten. Stream differential equations: concrete formats for coinductive definitions. Technical Report No. RR-11-10, Oxford University, 2011.

[21] D. Lucanu, E.I. Goriac, G. Caltais, and G. Rosu. CIRC: A behavioral verification tool based on circular coinduction. In A. Kurz, M. Lenisa, and A. Tarlecki, editors, *Proc. CALCO 2009*, volume 5728 of *LNCS*, pages 433–442. Springer, 2009.

[22] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[23] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[24] N. Moreira, D. Pereira, and S.M. de Sousa. Deciding regular expressions (in-)equivalence in Coq. In W. Kahl and T. Griffin, editors, *Proc. RAMiCS 2012*, volume 7560 of *LNCS*, pages 98–113. Springer, 2012.

[25] A. Okhotin. Conjunctive and boolean grammars: The true general case of the context-free grammars. *Computer Science Review*, 9:27–59, 2013.

[26] D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. TCS 1981*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.

[27] D. Pous and D. Sangiorgi. Enhancements of the bisimulation proof method. In *Advanced Topics in Bisimulation and Coinduction*, pages 233–289. Cambridge University Press, 2012.

[28] J. Rot, F. Bonchi, M. Bonsangue, D. Pous, J. Rutten, and A. Silva. Enhanced coalgebraic bisimulation. *To appear in MSCS*, 2014. Available at `http://www.liacs.nl/~jrot/up-to.pdf`.

[29] J. Rot, M.M. Bonsangue, and J.J.M.M. Rutten. Coinductive proof techniques for language equivalence. In A. H. Dediu, C. Martín-Vide, and B. Truthe, editors, *Proc. LATA 2013*, volume 7810 of *LNCS*, pages 480–492. Springer, 2013.

[30] J.J.M.M. Rutten. Automata and coinduction (an exercise in coalgebra). In *Proc. CONCUR 1998*, volume 1466 of *LNCS*, pages 194–218. Springer, 1998.

[31] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[32] J.J.M.M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theoretical Computer Science*, 308(1-3):1–53, 2003.

[33] D. Sangiorgi. On the bisimulation proof method. *Math. Struct. in Comp. Sci.*, 8(5):447–479, October 1998.

[34] D. Turi and G.D. Plotkin. Towards a mathematical operational semantics. In *Proc. LICS 1997*, pages 280–291. IEEE Computer Society, 1997.