Enhanced Coinduction

Rot, J.C.

Chapter 2

Coinduction for languages

The set of all languages over a given alphabet can be turned into an (infinite) deterministic automaton. The proof principle of *coinduction* asserts that two languages are equal if they are bisimilar in this automaton. Thus, to show equality of languages it suffices to construct a suitable bisimulation. Bisimilarity and coinduction are the basis of a practical proof method for language equality [Rut98a], which has, for example, been used in effective procedures for deciding language equivalence of regular expressions (e.g., [KN12, Rut98a, LGCR09, CS11]).

In the current chapter, we enhance this coinductive proof method using up-to techniques. These techniques allow to prove bisimilarity, and thus language equality, by means of *bisimulations up-to*, which are often smaller and easier to construct than actual bisimulations. We show how to apply up-to techniques through a number of examples, including new proofs of classical results such as Arden's rule [HU79].

The up-to techniques introduced in this chapter are particularly suitable for reasoning about operations and calculi on languages. To achieve a general picture of sound up-to techniques, we consider behavioural differential equations [Rut03], which give a syntactic format for specifying operations in terms of language derivatives. We show that bisimulation up-to can be used for reasoning about any operation defined in this format, and use this to prove properties of the shuffle operator and of languages defined by Boolean grammars.

Deterministic automata and their notion of bisimulation are instances of more general concepts from the theory of coalgebras. Indeed, the current exposition is based on the coalgebraic treatment of automata that was initiated in [Rut98a], and the main results of this chapter can be obtained by instantiating the abstract coalgebraic theory of coinduction up-to developed in subsequent chapters of this thesis. Thereby, the current chapter provides a concrete, self-contained introduction to coinduction and up-to techniques, that requires little background knowledge.

Outline. The next section contains preliminaries on languages and bisimulations. In Section 2.2, we motivate and introduce bisimulation up-to for regular opera-

tions. Then in Section 2.3, this is generalized to soundness results for operations given by behavioural differential equations, and applied to several other examples. Further, we give an equivalent, semantic characterization of the class of operations that are definable by behavioural differential equations. In Section 2.4, we treat simulation up-to, to reason about language inclusion. In Section 2.5, we discuss related work.

2.1 Bisimulations and coinduction

Throughout this chapter we assume a fixed alphabet A. The set of words is denoted by A^*, the empty word by ε and the concatenation of words w and v by wv. We let $2 = \{0, 1\}$ be the set of Boolean values, where $0 \leq 1$. A language is a set of words, which we represent as a function from A^* to 2; the set of languages is denoted by 2^{A^*}. We abuse notation and use 0 and 1 to denote the empty language and the language containing only the empty word respectively. Further we let any alphabet letter $a \in A$ denote the language that contains only the letter itself.

A *(deterministic) automaton (DA)* over A is a triple (X, o, t) where X is a set of states, $o\colon X \to 2$ is an output function, and $t\colon X \to X^A$ is a transition function. A state $x \in X$ is *final* or *accepting* if $o(x) = 1$. We do not require X to be finite and fix no initial states.

The classical definition of *bisimulation* applies to labelled transition systems, which, in contrast to deterministic automata, do not have output and may feature non-deterministic branching behaviour[1].

Definition 2.1.1. Let (X, o, t) be a deterministic automaton. A relation $R \subseteq X \times X$ is a *bisimulation* if for any $(x, y) \in R$:

1. $o(x) = o(y)$, and
2. for all $a \in A\colon (t(x)(a), t(y)(a)) \in R$.

The largest bisimulation is denoted by $\sim$ and called *bisimilarity*; if $x \sim y$ then we say x is *bisimilar* to y.

We will instantiate the notion of bisimulation to a special deterministic automaton, whose state space is given by the set of languages 2^{A^*}. The output of a language L is simply $L(\varepsilon)$, that is, a language is an accepting state precisely if it contains the empty word. For any $a \in A$, the a-transition from a language L is given by *language derivative* L_a, defined as follows for all $w \in A^*$:

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

[1] Bisimulation-like techniques have been used earlier in the setting of automata. In fact, the standard reference [Par81] introduces bisimulations for automata rather than transition systems, and Theorem 2.1.2 appears already there. For a historical account of bisimulation and coinduction, see [San12b].

Spelling out Definition 2.1.1, a relation R on languages is a bisimulation on this automaton if for any $(L, K) \in R$:

$$L(\varepsilon) = K(\varepsilon) \text{ and for all } a \in A\colon (L_a, K_a) \in R\,.$$

It turns out that bisimilarity of languages is a characterization of language equality. This is called the *coinductive proof principle*, or simply coinduction [Rut98a]. A more general account of coinduction is given in the next chapter.

Theorem 2.1.2 (Coinduction)**.** *For any two languages L, K:*

$$L \sim K \textit{ iff } L = K\,.$$

Proof. For the implication from right to left, one shows that the diagonal relation $\{(L, L) \mid L \in 2^{A^*}\}$ is a bisimulation. For the converse, one can prove that for any languages L, K and any word w: if $L \sim K$ then $L(w) = K(w)$, by (structural) induction on w. □

The above coinduction principle is a concrete proof method: to show that two languages L, K are equal, it suffices to construct a bisimulation containing (L, K).

2.1.1 Regular operations

Consider the regular operations of union $L + K$, concatenation $L \cdot K$ (often written as LK) and Kleene star L^*. These are defined, for all w, as usual: $(L + K)(w) = L(w) \vee K(w)$, $(L \cdot K)(w) = 1$ iff there are v, u such that $w = vu$ and $L(v) = 1 = K(u)$, and $L^* = \sum_{i \geq 0} L^i$, where $L^0 = 1$ and $L^{i+1} = L \cdot L^i$. In order to prove equivalence of expressions involving the above operations, we may use bisimulations, but this requires a characterization of the output (acceptance of the empty word) and the derivatives of operations in terms of their arguments. Such a characterization was given for regular expressions by Brzozowski [Brz64]; we formulate this in terms of languages (see, e.g., [Con71, page 41]).

Lemma 2.1.3. *For any two languages L, K and for any $a, b \in A$:*

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

Remark 2.1.4. This characterization in terms of output and derivative can equivalently be taken as the *definition* of the operations. This is achieved by constructing a deterministic automaton on the state space of *expressions* over languages. Such definition techniques, which are standard in the theory of coalgebras, are discussed in more detail in Chapter 3.

Example 2.1.5. Let $A = \{a, b\}$. We prove that $(a + b)^* = (a^*b^*)^*$. To this end, we start with the relation $R = \{((a + b)^*, (a^*b^*)^*)\}$ and try to show that 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 2.1.3, we see that $(a + b)^*(\varepsilon) = 1 = (a^*b^*)^*(\varepsilon)$. Moreover, again using Lemma 2.1.3, we have $((a + b)^*)_a = (a + b)_a(a + b)^* = (1 + 0)(a + b)^* = (a + b)^*$ and $((a^*b^*)^*)_a = (a^*b^*)_a(a^*b^*)^* = ((a^*)_a b^* + a^*(\varepsilon) \cdot (b^*)_a)(a^*b^*)^* = (a^*b^* + 0)(a^*b^*)^* = (a^*b^*)^*$, so the a-derivatives are again related (notice that apart from Lemma 2.1.3, we have used some basic facts about union and concatenation). The b-derivative of $(a + b)^*$ is $(a + b)^*$ itself; however, 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 Theorem 2.1.2.

Constructing a bisimulation (by hand) often follows the above pattern of using Lemma 2.1.3 to compute outputs and derivatives, extending the relation when necessary, and showing that the outputs are equal and the derivatives related. In the remainder of this chapter, we frequently use Lemma 2.1.3 without further reference to it.

If one restricts to regular languages, then the technique of constructing bisimulations in this manner gives rise to an effective algorithm for checking equivalence (cf. [Brz64, Con71, Rut98a]). However, the above coinductive proof method applies to equations over arbitrary languages, not only to regular ones, and in the next sections we consider many such instances of equations.

2.2 Bisimulation up-to for regular operations

In this section, we introduce an enhancement of the bisimulation proof method for equality of languages. We first illustrate the need for such an enhancement with a few examples. Consider the property $LL^* + 1 = L^*$. In order to prove this identity coinductively, we may try to show that

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

is a bisimulation. Using Lemma 2.1.3, it is easy to see that $(LL^* + 1)(\varepsilon) = L^*(\varepsilon)$ for any language L. Further, for any $a \in A$:

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

where the leftmost and rightmost equality are by Lemma 2.1.3, and in the second step we use some standard identities. Now we have shown that the derivatives are *equal*; this does not show that R is a bisimulation, since for that, the derivatives need to be *related by* R. The solution, however, is straightforward. If we augment

the relation R as follows:

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

then the derivatives of $LL^* + 1$ and L^* are related by R'; moreover, the diagonal is easily seen to satisfy the properties of a bisimulation as well. This solves the problem, but it is arguably somewhat inconvenient that additional work is required to deal with derivatives that are already equal.

As another motivating example, we consider the relation

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

The derivatives are (using Lemma 2.1.3):

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

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

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

Definition 2.2.1. For a relation $R \subseteq 2^{A^*} \times 2^{A^*}$, define the *congruence closure* of R (with respect to $+$, $\cdot$ and *) as the least relation $\equiv$ satisfying the following rules:

$$\frac{L\, R\, K}{L \equiv K} \qquad \frac{}{L \equiv L} \qquad \frac{L \equiv K}{K \equiv L} \qquad \frac{L \equiv K \qquad K \equiv M}{L \equiv M}$$

$$\frac{L_1 \equiv K_1 \qquad L_2 \equiv K_2}{L_1 + L_2 \equiv K_1 + K_2} \qquad \frac{L_1 \equiv K_1 \qquad L_2 \equiv K_2}{L_1 \cdot L_2 \equiv K_1 \cdot K_2} \qquad \frac{L \equiv K}{L^* \equiv K^*}$$

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

The first rule ensures that $R \subseteq \equiv_R$. The three rules on the right in the first row turn $\equiv_R$ into an equivalence relation. The three rules on the second row ensure that $\equiv_R$ is closed under the operations under consideration, which in particular means that $\equiv_R$ relates languages obtained by (syntactic) substitution of languages related by R. For example, if $(L^*L+1, L^*) \in R$, then we can derive from the above rules that $L_a(L^*L + 1) \equiv_R L_a L^*$.

Definition 2.2.2. A relation $R \subseteq 2^{A^*} \times 2^{A^*}$ is a *bisimulation up to congruence*, or simply a *bisimulation up-to*, if for any pair $(L, K) \in R$:

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

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

A bisimulation up-to is, in general, not a bisimulation. However, as we show below, it *represents* one, in the following sense: if R is a bisimulation up-to, then $\equiv_R$ is a bisimulation.

Theorem 2.2.3. *If R is a bisimulation up to congruence then for any $(L, K) \in R$, we have $L = K$.*

Proof. Let $\equiv$ be the congruence closure of R. We show that any pair (L, K) in $\equiv$ satisfies the properties

1. $L(\varepsilon) = K(\varepsilon)$ and
2. for any $a \in A$: $L_a \equiv K_a$

of a bisimulation, by rule induction on $(L, K) \in \equiv$. This amounts to showing that $\equiv$ is closed under the inference rules of Definition 2.2.2. 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 $L \equiv L$ is trivial.

Now assume languages L, K, M, N such that $L \equiv K$, $M \equiv N$, $L(\varepsilon) = K(\varepsilon)$, $M(\varepsilon) = N(\varepsilon)$ and for all $a \in A$: $L_a \equiv K_a$ and $M_a \equiv N_a$. We need to prove that $(L + M, K + N), (LM, KN), (L^*, K^*)$, (K, L) and (L, N) (if $K = M$) again satisfy the properties of a bisimulation, i.e., $(L + M)(\varepsilon) = (K + N)(\varepsilon)$ and for all $a \in A$: $(L + M)_a \equiv (K + N)_a$, and similarly for the other operations. We treat the case of union: $(L + M)(\varepsilon) = L(\varepsilon) \vee M(\varepsilon) = K(\varepsilon) \vee N(\varepsilon) = (K + N)(\varepsilon)$; moreover by assumption and closure of $\equiv$ under $+$ we have $L_a + M_a \equiv K_a + N_a$, and so $(L + M)_a = L_a + M_a \equiv K_a + N_a = (K + N)_a$.

Concatenation and Kleene star are treated in a similar manner, and symmetry and transitivity are not difficult either. Thus, by induction, $\equiv$ is a bisimulation, so by Theorem 2.1.2 we have $L = K$ for any $L \equiv K$ and for any $(L, K) \in R$, in particular. □

Any bisimulation is also a bisimulation up-to, so Theorem 2.2.3 is a generalization of Theorem 2.1.2 for the case of languages. Consequently, its converse holds as well.

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

Example 2.2.4. Recall the relation $R = \{(L^*L + 1, L^*) \mid L \in 2^{A^*}\}$ from the beginning of this section. As we have seen, the a-derivatives are $L_a(L^*L + 1)$ and L_aL^*, which are not related by R; however they *are* related by $\equiv_R$. So R is a bisimulation up-to, and consequently $L^*L + 1 = L^*$, by Theorem 2.2.3. Moreover, the relation $\{(LL^* + 1, L^*) \mid L \in 2^{A^*}\}$ from the beginning of this section is a bisimulation up-to as well; there, the derivatives are equal and thus related by $\equiv_R$.

Example 2.2.5. In order to prove $M + KL \subseteq L \Rightarrow K^*M \subseteq L$, we use that $K^*M \subseteq L$ if and only if $K^*M + L = L$, and try to prove that the relation

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

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

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

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

The above approach of dealing with language inclusion by reducing it to equality is, in general, not the most efficient one. In Section 2.4, we introduce *simulation up-to* which allows to deal with inequality more directly, and reprove the above example in a shorter way.

Example 2.2.6. Arden's rule, in a special case[2], states that if $L = KL + M$ for some languages L, K and M, and K does not contain the empty word, then $L = K^*M$. In order to prove its validity coinductively, let L, K, M be languages such that $K(\varepsilon) = 0$ and $L = KL + M$, and let $R = \{(L, K^*M)\}$. Using that $K(\varepsilon) = 0$, we have $L(\varepsilon) = (KL + M)(\varepsilon) = (K(\varepsilon) \wedge L(\varepsilon)) \vee M(\varepsilon) = (0 \wedge L(\varepsilon)) \vee M(\varepsilon) = M(\varepsilon) = 1 \wedge M(\varepsilon) = K^*(\varepsilon) \wedge M(\varepsilon) = K^*M(\varepsilon)$. Further,

$$\begin{aligned}
L_a &= (KL + M)_a = K_a L + K(\varepsilon) \cdot L_a + M_a \\
&= K_a L + M_a \equiv_R K_a K^*M + M_a = (K^*M)_a
\end{aligned}$$

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

While Arden's rule is not extremely difficult to prove without using bisimulations, the textbook proofs are longer and arguably more involved than the above proof, which is not much more than taking derivatives combined with a bit of algebraic reasoning, and does not require much ingenuity. Nevertheless, this coinductive proof is completely precise. Giving a formal proof without using these methods seems non-trivial; see [FS12] for the discussion of a proof within the theorem prover Isabelle.

[2]We consider a more general version of Arden's rule in Section 2.4.

In fact, [Rut98a] 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 [Rut98a] the infinite relation $\{(NL + O, NK^*M + O) \mid N, O \in 2^{A^*}\}$ is used, requiring more work in checking the bisimulation conditions. In that case, one essentially closes the relation $\{(L, K^*M)\}$ under (certain) contexts manually—this happens in a general and systematic fashion in the proof of Theorem 2.2.3.

Example 2.2.7. We prove that for any language L: $LL = 1 \Rightarrow L = 1$ (this property was used in [Koz90] to show that the universal Horn theory of Kleene algebra does not coincide with that of the regular sets). Assume $LL = 1$ and let $R = \{(L, 1)\}$. Since $(LL)(\varepsilon) = 1(\varepsilon) = 1$ also $L(\varepsilon) = 1 = 1(\varepsilon)$. We show that the derivatives of L and 1 are equal, turning R into a bisimulation up-to. First, for any $a \in A$: $L_aL + L_a = L_aL + L(\varepsilon)L_a = (LL)_a = 1_a = 0$. Now, one easily proves that this implies $L_a = 0$ (for example by showing that $\{(K, 0) \mid L + K = 0\}$ is a bisimulation). Thus $L_a = 0 = 1_a$, so $L_a \equiv_R 1_a$.

Example 2.2.8. We prove that $LL = L \Rightarrow L^* = 1 + L$, by establishing a bisimulation up-to (in fact, this example can also easily be proved by induction). To this end, let L be a language with $LL = L$ and consider the relation $R = \{(L^*, 1 + L)\}$. Indeed, $L^*(\varepsilon) = 1 = (1 + L)(\varepsilon)$, and for any $a \in A$: $(L^*)_a = L_aL^* \equiv_R L_a(1 + L) = L_a + L_aL = L_a + L(\varepsilon)L_a + L_aL = L_a + (LL)_a = L_a + L_a = L_a$.

The last example of this section concerns context-free languages. These can be expressed in terms of language equations [GR62]. For example, the language $\{a^nb^n \mid n \in \mathbb{N}\}$ is the unique language L such that $L = aLb + 1$.

Example 2.2.9. Let L, K, M be languages such that $L = aKMb+1$, $K = aMLb+1$ and $M = aLKb + 1$. Without thinking of what possible concrete descriptions of L, K and M can be, we show that $L = K = M$. To this end, let $R = \{(L, K), (K, M)\}$. Obviously $L(\varepsilon) = K(\varepsilon)$ and $K(\varepsilon) = M(\varepsilon)$. Moreover for any alphabet letter b other than a, we have $L_b = 0 = K_b$ and $K_b = 0 = M_b$. For the a-derivatives we have $L_a = KMb \equiv_R MKb \equiv_R MLb = K_a$ and similarly for (M, K); so R is a bisimulation up-to, proving that $L = K = M$.

2.3 Sound operations for bisimulation up-to

In the previous sections, we considered the regular operations on languages, and how their coinductive characterization can be used to prove equalities using bisimulations up-to. Next, we introduce a general syntactic format of operations on languages, and prove that a corresponding notion of bisimulation up-to is sound for any operation that can be characterized within this format. This format consists of a well-defined class of *behavioural differential equations* (BDEs) [Rut98a, Rut03]. More precisely, it is a variant of the *stream GSOS* format given in [HKR14] for stream systems. In fact, our format is a special case of *abstract GSOS* [TP97], a

categorical specification format at the level of coalgebras. In the following chapters, we recall abstract GSOS and prove soundness results for up-to techniques at this level, to obtain proof techniques not only for deterministic automata but for arbitrary coalgebras.

After introducing the general soundness results, we show several examples involving language equations with Boolean connectives, and the shuffle product. This section is concluded with a discussion of *causal* functions, which turn out to give a semantic characterization of operations that can be defined by behavioural differential equations [KNR11, HKR14].

A *signature* Σ is a countable set of operator names $\sigma \in \Sigma$ with associated arities[3] $|\sigma| \in \mathbb{N}$. A *language interpretation* of a signature Σ is a set of functions

$$\{\hat{\sigma} \colon (2^{A^*})^{|\sigma|} \to 2^{A^*}\}_{\sigma \in \Sigma}\,.$$

In the sequel, every language interpretation for a signature is of the above type (on languages), and so we simply speak about an *interpretation* and write $\{\hat{\sigma}\}_{\sigma\in\Sigma}$.

Definition 2.3.1. For a relation $R \subseteq 2^{A^*} \times 2^{A^*}$, define the *congruence closure* $\equiv_R^\Sigma$ of R w.r.t. an interpretation $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ as the least relation $\equiv$ satisfying the following rules:

$$\frac{L\,R\,K}{L \equiv K} \qquad \frac{}{L \equiv L} \qquad \frac{L \equiv K}{K \equiv L} \qquad \frac{L \equiv K \qquad K \equiv M}{L \equiv M}$$

$$\frac{L_1 \equiv K_1 \qquad \dots \qquad L_n \equiv K_n}{\hat{\sigma}(L_1, \dots, L_n) \equiv \hat{\sigma}(K_1, \dots, K_n)} \quad \text{for each } \sigma \in \Sigma, n = |\sigma|$$

R is a *bisimulation up-to (w.r.t. $\{\hat{\sigma}\}_{\sigma\in\Sigma}$)*, if for any $(L, K) \in R$:

1. $L(\varepsilon) = K(\varepsilon)$, and
2. for all $a \in A \colon L_a \equiv_R^\Sigma K_a$.

Bisimulation up-to for the regular operators (Definition 2.2.1) is a special case of the above definition. While Theorem 2.2.3 asserts that bisimulation up-to is a sound proof technique in the case of union, concatenation and Kleene star, this is not the case in general for other operations. This is illustrated by the following example, adapted from [PS12].

Example 2.3.2. Assume for simplicity a singleton alphabet $\{a\}$. Consider the signature that only contains a unary operator h, whose interpretation is defined as follows:

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

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

[3] 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 introduce a condition that guarantees soundness, based on characterizing the operations in terms of BDEs [Rut03]. 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 2.1.3 form an example. Indeed, behavioural differential equations are best explained through concrete examples. To prove our soundness theorem, however, we need a precise characterization.

Define the set of *terms* $\Sigma^*(V)$ over a signature Σ and a set of variables V by the grammar

$$t ::= v \mid \sigma(t_1, \ldots, t_n)$$

where v ranges over V, σ ranges over Σ and $n = |\sigma|$. Given an interpretation $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ we define a function

$$I\colon \Sigma^*(2^{A^*}) \to 2^{A^*}$$

by induction: $I(L) = L$ and $I(\sigma(t_1, \ldots, t_n)) = \hat{\sigma}(I(t_1), \ldots, I(t_n))$. Substitution in t of a term u for a variable x is denoted by $t[x := u]$.

Definition 2.3.3. A *(syntactic) behavioural differential equation (BDE)* over a signature Σ for an operator $\sigma \in \Sigma$ of arity n consists of a pair (o, d) of functions of the form

$$o\colon 2^n \to 2 \qquad \text{and} \qquad d\colon A \to \Sigma^*(V_n)$$

where V_n is a set consisting of variables

- $x_1, \ldots, x_n$,
- $x_1^\varepsilon, \ldots, x_n^\varepsilon$ and
- for each $a \in A$ and each $i \leq n$ a variable x_i^a,

all of which are pairwise distinct.

The function o specifies the output of the operation given the output of the arguments, and the function d specifies, for each alphabet letter, the derivative. This derivative is given as a term; intuitively a variable x_i represents the i-th argument of the operation, a variable x_i^ε represents its output, and a variable x_i^a represents the a-derivative of the i-th argument. For instance, the equations for language concatenation in Lemma 2.1.3 would be presented as a behavioural differential equation (o, d), where $o\colon 2 \times 2 \to 2$ is conjunction, and $d(a) = x_1^a \cdot x_2 + x_1^\varepsilon \cdot x_2^a$ for all $a \in A$.

To formalize the intuition that syntactic behavioural differential equations define actual equations on languages, we define for each n a function

$$\begin{aligned} &\rho_n\colon \Sigma^*(V_n) \to ((2^{A^*})^n \to \Sigma^*(2^{A^*})) \\ &\rho_n(t)(L_1, \ldots, L_n) = t[x_i := L_i]_{i\leq n}[x_i^a := (L_i)_a]_{i\leq n, a\in A}[x_i^\varepsilon := L_i(\varepsilon)]_{i\leq n} \end{aligned} \tag{2.1}$$

which substitutes each x_i by L_i, x_i^a by the a-derivative $(L_i)_a$ and x_i^ε by $L(\varepsilon)$. Now, given a function $\hat{\sigma}\colon (2^{A^*}) \to 2^{A^*}$, a BDE (o, d) for σ defines an *equation* for each $a \in A$:

$$\hat{\sigma}(L_1, \ldots, L_n)_a = I(\rho_n(d(a))(L_1, \ldots, L_n)) \tag{2.2}$$

which states in a precise manner that the a-derivative of $\hat{\sigma}(L_1, \ldots, L_n)$ behaves according to the syntactic presentation $d(a)$. For instance, if $\hat{\sigma}$ is language composition and $d(a) = x_1^a \cdot x_2 + x_1^\varepsilon \cdot x_2^a$ then the equation corresponds to

$$(L_1 \cdot L_2)_a = (L_1)_a \cdot L_2 + L_1(\varepsilon) \cdot (L_2)_a \,.$$

If, for an arbitrary operation $\hat{\sigma}$ and BDE (o, d) the equation 2.2 holds and, moreover, the output of $\hat{\sigma}(L_1, \ldots, L_n)$ is given by o applied to the output of its arguments, then we say $\hat{\sigma}$ is *given by* (o, d). This is captured formally by the following definition.

Definition 2.3.4. We say an interpretation $\{\hat{\sigma}\}_{\sigma \in \Sigma}$ *can be given by BDEs* if for each σ (with arity n) there is a BDE (o, d) over Σ such that for all languages $L_1, \ldots, L_n$:

$$\begin{aligned} \hat{\sigma}(L_1, \ldots, L_n)(\varepsilon) &= o(L_1(\varepsilon), \ldots, L_n(\varepsilon)) \\ \hat{\sigma}(L_1, \ldots, L_n)_a &= I(\rho_n(d(a))(L_1, \ldots, L_n)) \end{aligned}$$

where ρ^n is defined as in (2.1).

Remark 2.3.5. A behavioural differential equation (o, d) as in Definition 2.3.3 induces for each set X a function

$$d'_X\colon A \to (X^n \times (X^A)^n \times 2^n \to \Sigma^*(X))$$

which is *natural* in X, informally meaning that $d'_X(a)$ is defined uniformly over every set X. This view allows for a neater formalization of presentation by BDEs (Definition 2.3.4) with respect to operations on an *arbitrary* deterministic automaton, which concides with Definition 2.3.4 for the case of the automaton of languages. Since we aim here to use as few technical notions as possible we postpone such a treatment to Section 3.5. There, we recall a general approach to define and study operations and calculi based on the theory of algebras and coalgebras, with behavioural differential equations as a special case.

Lemma 2.1.3 states that the regular operations are captured by BDEs. So the following theorem generalizes the proof principle of Theorem 2.2.3.

Theorem 2.3.6. *If $\{\hat{\sigma}\}_{\sigma \in \Sigma}$ can be given by BDEs, then for any relation R which is a bisimulation up-to w.r.t. $\{\hat{\sigma}\}_{\sigma \in \Sigma}$: if $(L, K) \in R$ then $L = K$.*

Proof. Similarly to the proof of Theorem 2.2.3, we show that the congruence closure $\equiv$ of R is a bisimulation, by proving by rule induction that

- $L(\varepsilon) = K(\varepsilon)$ and

- for any $a \in A$: $L_a \equiv K_a$

holds for any $(L, K) \in \equiv$. The base cases, i.e., if $L = K$ or $(L, K) \in R$, are the same as in Theorem 2.2.3.

The rules for symmetry and transitivity are not difficult. We treat the rule for an operator $\sigma \in \Sigma$ of arity $n = |\sigma|$. Let o and d be the functions from Definition 2.3.3 associated to σ which exist since $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ can be given by BDEs, and suppose we have languages $L_1, \ldots, L_n$ and $K_1, \ldots, K_n$ such that for all i:

$$L_i \equiv K_i,\ L_i(\varepsilon) = K_i(\varepsilon) \text{ and for all } a \in A\colon (L_i)_a \equiv (K_i)_a\,. \tag{2.3}$$

Then we have

$$\begin{aligned}\hat{\sigma}(L_1, \ldots, L_n)(\varepsilon) &= o(L_1(\varepsilon), \ldots, L_n(\varepsilon))\\ &= o(K_1(\varepsilon), \ldots, K_n(\varepsilon)) = \hat{\sigma}(K_1, \ldots, K_n)(\varepsilon)\end{aligned}$$

and for any $a \in A$:

$$\begin{aligned}\hat{\sigma}(L_1, \ldots, L_n)_a &= I(\rho_n(d(a))(L_1, \ldots, L_n))\\ &= I(d(a)[x_i := L_i]_{i\le n}[x_i^a := (L_i)_a]_{i\le n, a\in A}[x_i^\varepsilon := L_i(\varepsilon)]_{i\le n})\\ &\equiv I(d(a)[x_i := K_i]_{i\le n}[x_i^a := (K_i)_a]_{i\le n, a\in A}[x_i^\varepsilon := K_i(\varepsilon)]_{i\le n})\\ &= I(\rho_n(d(a))(K_1, \ldots, K_n))\\ &= \hat{\sigma}(K_1, \ldots, K_n)_a\end{aligned}$$

where the third step (relation by $\equiv$) holds by the induction hypothesis (2.3). □

Bisimulation up-to with respect to the function $\hat{h}$ of Example 2.3.2 is not sound, as we have seen. Indeed $\hat{h}$ cannot be given by BDEs, since the output $\hat{h}(L)(\varepsilon)$ depends not only on $L(\varepsilon)$ but on the entire language L.

2.3.1 Language equations with complement and intersection

Language complement $\overline{L}$ and intersection $L \wedge K$ are defined as $\overline{L}(w) = \neg(L(w))$ and $(L \wedge K)(w) = L(w) \wedge K(w)$ respectively. Language equations including these additional operations can be used to give semantics to *conjunctive* and *Boolean grammars*, which extend context-free grammars with conjunction and complement [Okh13]. Complement and intersection have a known characterization in terms of outputs and derivatives as well [Brz64]:

Lemma 2.3.7. *For any two languages L, K and for any $a \in A$:*

$$\begin{aligned}\overline{L}(\varepsilon) &= \neg(L(\varepsilon)) & (\overline{L})_a &= \overline{(L_a)}\\ (L \wedge K)(\varepsilon) &= L(\varepsilon) \wedge K(\varepsilon) & (L \wedge K)_a &= L_a \wedge K_a\end{aligned}$$

The above characterization is in terms of BDEs, so by Theorem 2.3.6 we obtain the soundness of bisimulation up-to.

Example 2.3.8. There are unique languages L and K such that

$$L = aLa + bLb + a + b + 1 \qquad K = aKa + bKb + aA^*b + bA^*a$$

L is the language of *palindromes*, i.e., words which are equal to their own reverse. We claim that K is the language of all *non*-palindromes, and prove this formally by showing that the relation $R = \{(\overline{L}, K)\}$ is a bisimulation up-to. The outputs are easily seen to be equal: $\overline{L}(\varepsilon) = \neg(L(\varepsilon)) = \neg(1(\varepsilon)) = 0 = K(\varepsilon)$.

$$\begin{aligned}\overline{L}_a &= \overline{L_a} = \overline{La + 1} = \overline{La} \wedge \overline{1} = (\overline{L}a + A^*b + 1) \wedge \overline{1} \\ &\equiv_R (Ka + A^*b + 1) \wedge \overline{1} = Ka \wedge \overline{1} + A^*b \wedge \overline{1} + 1 \wedge \overline{1} = Ka + A^*b = K_a\end{aligned}$$

In the fourth step, we unfold the complement $\overline{La}$, the validity of which is itself a nice exercise in bisimulation up-to. Further, the case of b-derivatives is of course similar to the above. So R is a bisimulation up-to, proving that K indeed is the complement of L.

2.3.2 Shuffle (closure)

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

$$L^{\circledast} = \sum_{i=0}^{\infty} L^{\otimes_i}$$

where $L^{\otimes_i}$ is defined inductively by $L^{\otimes_0} = 1$ and $L^{\otimes_{i+1}} = L \otimes L^{\otimes_i}$. Notice that the shuffle closure is 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 BDEs, as stated by the following lemma.

Lemma 2.3.9. *For any two languages L, K and for any $a, b \in A$:*

$$\begin{aligned}(L \otimes K)(\varepsilon) &= L(\varepsilon) \wedge K(\varepsilon) & (L \otimes K)_a &= L_a \otimes K + L \otimes K_a \\ L^{\circledast}(\varepsilon) &= 1 & (L^{\circledast})_a &= L_a \otimes L^{\circledast}\end{aligned}$$

As an example of a proof using bisimulation up-to that involves the shuffle operator, we treat the unfolding of the shuffle closure.

Example 2.3.10. Let L be any language; then $L^{\circledast} = L \otimes L^{\circledast} + 1$. To show this, let $R = \{(L^{\circledast}, L \otimes L^{\circledast} + 1\}$. Then $L^{\circledast}(\varepsilon) = 1 = (L \otimes L^{\circledast} + 1)(\varepsilon)$. Moreover for any alphabet letter a:

$$\begin{aligned}(L^{\circledast})_a &= L_a \otimes L^{\circledast} = L_a \otimes (L^{\circledast} + L^{\circledast}) \\ &\equiv_R L_a \otimes (L^{\circledast} + L \otimes L^{\circledast} + 1) = L_a \otimes (L^{\circledast} + L \otimes L^{\circledast}) \\ &= L_a \otimes L^{\circledast} + L_a \otimes L \otimes L^{\circledast} = L_a \otimes L^{\circledast} + L \otimes L_a \otimes L^{\circledast} \\ &= (L \otimes L^{\circledast} + 1)_a\end{aligned}$$

using Lemma 2.3.9, Lemma 2.1.3 and some basic identities. Thus R is a bisimulation up-to, proving $L^{\circledast} = L \otimes L^{\circledast} + 1$.

2.3.3 Causal functions

The format of BDEs defined in this section is a straightforward extension of the one for streams, given in [KNR11, HKR14]. There, it is shown that functions that can be given by BDEs are exactly those that are *causal*, and vice versa. This result can be extended to the case of languages. As a consequence of Theorem 2.3.6, we then obtain causality of functions as an equivalent, *semantic* condition for soundness of up-to techniques. Here, we assume the alphabet A to be finite.

For any language L and any $k \in \mathbb{N}$ we define $L|_k \in 2^{A^*}$ by

$$L|_k(w) = \begin{cases} L(w) & \text{if } |w| \leq k \\ 0 & \text{otherwise} \end{cases}$$

where $|w|$ is the length of a word w. Define the relation $\approx_k$ as follows:

$$L \approx_k K \text{ iff } L|_k = K|_k \,.$$

A function $\hat{\sigma} \colon (2^{A^*})^n \to 2^{A^*}$ is *causal* if for all languages $L_1, \ldots, L_n$, $K_1, \ldots, K_n$ and for any $k \in \mathbb{N}$:

$$L_1 \approx_k K_1, \ldots, L_n \approx_k K_n \quad \text{implies} \quad \hat{\sigma}(L_1, \ldots, L_n) \approx_k \hat{\sigma}(K_1, \ldots, K_n) \,.$$

Causality means that equality up to length k is a congruence, for any k. In other words, membership in $\hat{\sigma}(L_1, \ldots, L_n)$ of words of length less than k depends only on the words in $L_1, \ldots, L_n$ of length less than k. For example, the function $\hat{h}$ from Example 2.3.2 is not causal: whether or not $\hat{h}(L)$ contains the empty word depends on the entire language L.

Lemma 2.3.11. *The set of all causal functions can be given by BDEs.*

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 $\hat{\sigma} \colon 2^{A^*} \to 2^{A^*}$; 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

$$\tilde{\sigma}_a \colon (2^{A^*})^{l+1} \to 2^{A^*}$$

defined as $\tilde{\sigma}_a(M, K_1, \ldots, K_l) = (\hat{\sigma}(M(\varepsilon) + a_1 K_1 + \ldots + a_l K_l))_a$. Then $\tilde{\sigma}_a$ is causal, and it follows that

$$\hat{\sigma}(L)_a = \tilde{\sigma}_a(L(\varepsilon), L_{a_1}, \ldots, L_{a_l}) \,.$$

We have thus expressed the derivative $\hat{\sigma}(L)_a$ in terms of another causal function, which takes the output and derivatives of L as arguments. Further, since $\hat{\sigma}$ is causal, the output $\hat{\sigma}(L)(\varepsilon)$ depends only on $L(\varepsilon)$. □

In order to prove the converse, that is, any operation that can be given by BDEs is causal, we need the following.

Lemma 2.3.12. *Let $k \in \mathbb{N}$. Suppose that for all $\hat{\sigma}$ in some set $\{\hat{\sigma}\}_{\sigma\in\Sigma}$, and for all languages $L_1, \ldots, L_n, K_1, \ldots, K_n$ (where $n = |\sigma|$) we have*

$$L_1 \approx_k K_1, \ldots, L_n \approx_k K_n \quad \textit{implies} \quad \hat{\sigma}(L_1, \ldots, L_n) \approx_k \hat{\sigma}(K_1, \ldots, K_n)\,. \tag{2.4}$$

Then for any list of variables $x_1, \ldots, x_m$, any term $t \in \Sigma^*(x_1, \ldots, x_m)$ over operators in Σ, and any languages $L_1, \ldots, L_m$:

$$L_1 \approx_k K_1, \ldots, L_m \approx_k K_m \quad \textit{implies} \quad I(t[x_i := L_i]_{i\leq m}) \approx_k I(t[x_i := K_i]_{i\leq m})\,.$$

Proof. Let $L_1, \ldots, L_n, K_1, \ldots, K_n$ be languages such that $L_1 \approx_k K_1, \ldots, L_m \approx_k K_m$, and suppose that (2.4) holds. We prove that $I(t[x_i := L_i]_{i\leq m}) \approx_k I(t[x_i := K_i]_{i\leq m})$ by structural induction on t.

For the base case, if t is a variable x_j then $I(t[x_i := L_i]_{i\leq m}]) = L_j$ and $I(t[x_i := K_i]_{i\leq m}) = K_j$, and we need to prove $L_j \approx_k K_j$ which trivially follows from our assumption.

Suppose $I(t_j[x_i := L_i]_{i\leq m}) \approx_k I(t_j[x_i := K_i]_{i\leq m})$ for all $j \leq n$. Then

$$\begin{aligned}
&I(\sigma(t_1, \ldots, t_n)[x_i := L_i]_{i\leq m}) \\
&= I(\sigma(t_1[x_i := L_i]_{i\leq m}, \ldots, t_n[x_i := L_i]_{i\leq m})) \\
&= \hat{\sigma}(I(t_1[x_i := L_i]_{i\leq m}), \ldots, I(t_n[x_i := L_i]_{i\leq m})) \\
&\approx_k \hat{\sigma}(I(t_1[x_i := K_i]_{i\leq m}), \ldots, I(t_n[x_i := K_i]_{i\leq m})) \\
&= I(\sigma(t_1[x_i := K_i]_{i\leq m}, \ldots, t_n[x_i := K_i]_{i\leq m})) \\
&= I(\sigma(t_1, \ldots, t_n)[x_i := K_i]_{i\leq m})
\end{aligned}$$

where the second step (relating by $\approx_k$) follows by the induction hypothesis and the assumption (2.4). □

Theorem 2.3.13. *A function $\hat{\sigma} \colon (2^{A^*})^n \to 2^{A^*}$ is causal if and only if it is contained in a set of functions which can be given by BDEs.*

Proof. From left to right, the result follows from Lemma 2.3.11. For the other direction, assume a set of functions given by BDEs. We prove that

$$L_1 \approx_k K_1, \ldots, L_n \approx_k K_n \quad \text{implies} \quad \hat{\sigma}(L_1, \ldots, L_n) \approx_k \hat{\sigma}(K_1, \ldots, K_n) \tag{2.5}$$

for every $\hat{\sigma}$ (with n the arity of $\hat{\sigma}$) in the set and for every k, by induction on k.

Take any $\hat{\sigma}$, with arity n, and given by the BDE (o, d). The base case ($k = 0$) holds since $L_1 \approx_0 K_1$, …, $L_n \approx_0 K_n$ implies

$$\begin{aligned}
\hat{\sigma}(L_1, \ldots, L_n)(\varepsilon) &= o(L_1(\varepsilon), \ldots, L_n(\varepsilon)) \\
&= o(K_1(\varepsilon), \ldots, K_n(\varepsilon)) = \hat{\sigma}(K_1, \ldots, K_n)(\varepsilon)\,.
\end{aligned}$$

Now suppose (2.5) holds for some $k \in \mathbb{N}$, and suppose that we have languages $L_1, \ldots, L_n, K_1, \ldots, K_n$ such that $L_i \approx_{k+1} K_i$ for each $i \leq n$. We need to prove:

$$\hat{\sigma}(L_1, \ldots, L_n) \approx_{k+1} \hat{\sigma}(K_1, \ldots, K_n)\,. \tag{2.6}$$

Since $L_i \approx_{k+1} K_i$ by assumption, we have for each $i \leq n$: $L_i \approx_k K_i$, $L_i(\varepsilon) \approx_k K_i(\varepsilon)$ and for each $a \in A$: $(L_i)_a \approx_k (K_i)_a$. By the induction hypothesis (2.5) and Lemma 2.3.12 it follows that, for each $a \in A$:

$$\begin{aligned} &I(d(a)[x_i := L_i]_{i\leq n}[x_i^a := (L_i)_a]_{i\leq n, a\in A}[x_i^\varepsilon := L_i(\varepsilon)]_{i\leq n}) \\ &\approx_k I(d(a)[x_i := K_i]_{i\leq n}[x_i^a := (K_i)_a]_{i\leq n, a\in A}[x_i^\varepsilon := K_i(\varepsilon)]_{i\leq n}) \end{aligned}$$

where d is the function that specifies the derivative of $\hat{\sigma}$, and thus

$$\begin{aligned} \hat{\sigma}(L_1, \ldots, L_n)_a &= I(\rho_n(d(a))(L_1, \ldots, L_n)) \\ &= I(d(a)[x_i := L_i]_{i\leq n}[x_i^a := (L_i)_a]_{i\leq n, a\in A}[x_i^\varepsilon := L_i(\varepsilon)]_{i\leq n}) \\ &\approx_k I(d(a)[x_i := K_i]_{i\leq n}[x_i^a := (K_i)_a]_{i\leq n, a\in A}[x_i^\varepsilon := K_i(\varepsilon)]_{i\leq n}) \\ &= I(\rho_n(d(a))(K_1, \ldots, K_n)) \\ &= \hat{\sigma}(K_1, \ldots, K_n)_a \,. \end{aligned}$$

Since, moreover, $\hat{\sigma}(L_1, \ldots, L_n)(\varepsilon) = \hat{\sigma}(K_1, \ldots, K_n)(\varepsilon)$ we get (2.6) as desired. □

By Theorem 2.3.6 and the above result we directly obtain causality as a sufficient condition for the soundness of bisimulation up-to.

Corollary 2.3.14. *Suppose every function of an interpretation $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ is causal. Then bisimulation up-to w.r.t. $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ is sound, i.e., if R is a bisimulation up-to w.r.t. $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ then $(L, K) \in R$ implies $L = K$.*

2.4 Simulation (up-to)

So far we have focused on techniques for showing equality of languages. Of course, these methods can also be used to prove language inclusion, since $L \subseteq K$ iff $L + K = K$. However, there is a more direct way: instead of bisimulations, one can construct *simulations*, which in practice turns out to be easier for proving language inclusion.

Definition 2.4.1. Let (X, o, t) be a deterministic automaton. A *simulation* is a relation $R \subseteq X \times X$ such that for any $(x, y) \in R$:

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

The difference with bisimulation is that condition (1) is relaxed: if x is a final state then y should be final as well, but if x is not final then the output of y does not matter.

Theorem 2.4.2. *If $R \subseteq 2^{A^*} \times 2^{A^*}$ is a simulation (on the automaton of languages defined in Section 2.1) then for any $(L, K) \in R$: $L \subseteq K$.*

Thus simulation is a concrete proof principle for language inclusion, just like bisimulation is a proof principle for language equality.

Simulation up-to is based on a *precongruence* rather than a congruence closure; the difference is that the precongruence is not symmetric, and it relates L to K whenever L is included in K.

Definition 2.4.3. For a relation $R \subseteq 2^{A^*} \times 2^{A^*}$, define the *precongruence closure* $\leqq_R^\Sigma$ of R w.r.t. an interpretation $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ as the least relation $\leqq$ satisfying the following rules:

$$\frac{L\,R\,K}{L \leqq K} \qquad \frac{L \subseteq K}{L \leqq K} \qquad \frac{L \leqq K \qquad K \leqq M}{L \leqq M}$$

$$\frac{L_1 \leqq K_1 \qquad \dots \qquad L_n \leqq K_n}{\hat{\sigma}(L_1,\dots,L_n) \leqq \hat{\sigma}(K_1,\dots,K_n)} \qquad \text{for each } \sigma \in \Sigma, n = |\sigma|$$

R is a *simulation up-to (w.r.t. an interpretation $\{\hat{\sigma}\}_{\sigma\in\Sigma}$)*, if for any $(L, K) \in R$:

1. $L(\varepsilon) \leq K(\varepsilon)$, and
2. for all $a \in A$: $L_a \leqq_R^\Sigma K_a$.

Our soundness criterion for bisimulation up-to, which is that the operations can be given by BDEs, turns out not to be strong enough for simulation up-to, as witnessed by the following example.

Example 2.4.4. The complement operation can be given by BDEs (Lemma 2.3.7). Consider the relation $R = \{(aA^*, 0)\}$. We have $(aA^*)(\varepsilon) = 0 = 0(\varepsilon)$. Moreover $(aA^*)_a = A^* = \overline{0}$ and $0_a = 0 = \overline{A^*}$. Since $0 \subseteq A^*$, we have $\overline{0} \leqq_R \overline{A^*}$ and thus $(aA^*)_a \leqq_R 0_a$, showing that R is a simulation up-to. But clearly $aA^* \not\subseteq 0$, so simulation up-to with respect to language complement is not a sound proof principle.

Our solution is to require in addition that the operations under consideration satisfy a monotonicity condition.

Definition 2.4.5. A set $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ of operations is given by *monotone BDEs* if

1. $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ is given by BDEs, and
2. for each $\sigma \in \Sigma$: the associated (output) function $o\colon 2^n \to 2$ is monotone, i.e., if $o_j \leq u_j$ for all j with $1 \leq j \leq n$ then $o(o_1,\dots,o_n) \leq o(u_1,\dots,u_n)$.

Theorem 2.4.6. *If $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ can be given by monotone BDEs then for any relation R which is a simulation up-to w.r.t. $\{\hat{\sigma}\}_{\sigma\in\Sigma}$: if $(L, K) \in R$ then $L \subseteq K$.*

Proof. The proof is mostly that of Theorem 2.3.6. 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 $\hat{\sigma}$ is an operation of arity n, from a set $\{\hat{\sigma}\}_{\sigma\in\Sigma}$ of operations given by monotone BDEs, and let o be

its output function. Let $L_1, \ldots, L_n$ and $K_1, \ldots K_n$ be languages such that for all j: $L_j(\varepsilon) \leq K_j(\varepsilon)$. Then

$$\begin{aligned}\hat{\sigma}(L_1, \ldots, L_n)(\varepsilon) &= o(L_1(\varepsilon), \ldots, L_n(\varepsilon)) \\ &\leq o(K_1(\varepsilon), \ldots, K_n(\varepsilon)) = \hat{\sigma}(K_1, \ldots, K_n)(\varepsilon)\end{aligned}$$

where we use the assumption that o is monotone. □

Example 2.4.7. The general version of Arden's rule states that, given languages K and M, the *least* solution of $L = KL + M$ is K^*M. Furthermore, if $K(\varepsilon) = 0$ then it is the unique one, as we have seen in Example 2.2.6. For the proof of the general statement, first notice that K^*M is indeed a solution since $K^*M = (KK^* + 1)M = KK^*M + M$. To show that it is the least one, let L be any language such that $L = KL + M$ and consider the relation $R = \{(K^*M, L)\}$. Then R is a simulation up-to, since $(K^*M)(\varepsilon) = M(\varepsilon) \leq (KL + M)(\varepsilon) = L(\varepsilon)$ and for any a:

$$\begin{aligned}(K^*M)_a &= K_aK^*M + M_a \leqq_R K_aL + M_a \\ &\subseteq K_aL + K(\varepsilon)L_a + M_a = (KL + M)_a = L_a\,.\end{aligned}$$

Thus K^*M is the least solution.

The reader is invited to formulate and prove a version of Arden's rule, where shuffle and shuffle closure (Section 2.3.2) replace concatenation and Kleene star. Further, in Example 2.2.5 we proved $M + KL \subseteq L \Rightarrow K^*M \subseteq L$ using a bisimulation up-to. The proof using a simulation up-to is the same as (part of) the above proof of Arden's rule. One might expect that $M + LK \subseteq L \Rightarrow MK^* \subseteq L$ has a similar treatment, but due to the asymmetry of the derivative of concatenation the proof is different.

Example 2.4.8. In order to prove $M + LK \subseteq L \Rightarrow MK^* \subseteq L$, consider the relation $R = \{(MK^*, L) \mid M + LK \subseteq L;\ L, K, M \in 2^{A^*}\}$. Let L, K, M be such languages; then $M(\varepsilon) \leq L(\varepsilon)$, so $(MK^*)(\varepsilon) \leq L(\varepsilon)$. For any $a \in A$, we have

$$(MK^*)_a = M_aK^* + M(\varepsilon)K_aK^* = (M_a + M(\varepsilon)K_a)K^*$$

In order to see that this is related by $\leqq_R$ to L_a, we start with our assumption $M + LK \subseteq L$ and compute derivatives: $(M + LK)_a \subseteq L_a$, so $M_a + L_aK + L(\varepsilon)K_a \subseteq L_a$. Reformulating this as $(M_a + L(\varepsilon)K_a) + L_aK \subseteq L_a$, we have

$$((M_a + L(\varepsilon)K_a)K^*, L_a) \in R\,.$$

Since $M(\varepsilon) \leq L(\varepsilon)$ we thus obtain

$$(MK^*)_a = (M_a + M(\varepsilon)K_a)K^* \subseteq (M_a + L(\varepsilon)K_a)K^* \leqq_R L_a$$

as desired, showing that R is a simulation up-to.

We conclude with the soundness of an axiom that concerns the interplay between shuffle and concatenation and that is used, for example, in concurrency theory [HMSW11].

Example 2.4.9. The *exchange law* states that

$$(M \otimes L)(K \otimes N) \subseteq (MK) \otimes (LN)$$

for any languages M, L, K, N. Consider the relation

$$R = \{((M \otimes L)(K \otimes N), (MK) \otimes (LN)) \mid M, K, L, N \in 2^{A^*}\}.$$

Then

$$((M \otimes L)(K \otimes N))(\varepsilon) = M(\varepsilon) \wedge L(\varepsilon) \wedge K(\varepsilon) \wedge N(\varepsilon) = ((MK) \otimes (LN))(\varepsilon)$$

and for any alphabet letter a:

$$\begin{aligned}
&((M \otimes L)(K \otimes N))_a \\
&= (M_a \otimes L + M \otimes L_a)(K \otimes N) + (M \otimes L)(\varepsilon)(K_a \otimes N + K \otimes N_a) \\
&= (M_a \otimes L)(K \otimes N) + (M \otimes L_a)(K \otimes N) + (M(\varepsilon) \wedge L(\varepsilon))(K_a \otimes N) \\
&\qquad + (M(\varepsilon) \wedge L(\varepsilon))(K \otimes N_a) \\
&\leqq_R (M_a K) \otimes (LN) + (MK) \otimes (L_a N) \\
&\qquad + (M(\varepsilon)K_a) \otimes (L(\varepsilon)N) + (M(\varepsilon)K) \otimes (L(\varepsilon)N_a) \\
&\subseteq (M_a K) \otimes (LN) + (MK) \otimes (L_a N) \\
&\qquad + (M(\varepsilon)K_a) \otimes (LN) + (MK) \otimes (L(\varepsilon)N_a) \\
&= (M_a K + M(\varepsilon)K_a) \otimes (LN) + (MK) \otimes (L_a N + L(\varepsilon)N_a) \\
&= ((MK) \otimes (LN))_a
\end{aligned}$$

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

The proof in the above example is clearly easier than one where the inclusion would be reduced to checking equality by means of bisimilarity.

2.5 Discussion and related work

We presented *bisimulation up-to* as a proof method for language equivalence, and *simulation up-to* as a proof method for language inclusion. These techniques are sound enhancements of the coinductive proof method based on (bi)simulation, if the operations under consideration adhere to the format of behavioural differential equations presented in this chapter. For the soundness of simulation up-to, the operations additionally need to satisfy a monotonicity condition.

Deterministic automata are coalgebras, and the notions of bisimulation and coinduction introduced in Section 2.1 are instances of general definitions [Rut98a]. The up-to techniques introduced in this chapter are also instances of much more general results developed in subsequent chapters of this thesis. In fact, the format of BDEs can be represented by a distributive law, which immediately proves the soundness (actually, a stronger notion) of bisimulation up-to. Moreover, the monotonicity condition for simulation up-to arises from a result in Chapter 5 that requires that the distributive law lifts to a certain category.

A discussion of related work with respect to more general up-to techniques is postponed to Chapter 5. Relevant in the present context is the work of Bonchi and Pous [BP13], which consists of a new algorithm for checking equivalence of non-deterministic automata based on bisimulation up to congruence, improving the state of the art significantly (see also [HR15]). That algorithm is based on the algebraic structure of the powerset of states, 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, in order to prove, e.g., inequalities over arbitrary languages.

Bisimulation up-to techniques have been applied to facilitate coinductive definitions and proofs in Coq [EHB13]. In fact, the latter paper uses causal contexts on streams as a condition for soundness; as shown in [HKR14] (and extended to languages in this chapter), this condition is equivalent to requiring that the operations under consideration can be defined by behavioural differential equations.

Our techniques are more widely applicable than only to regular languages, as we have shown in a number of examples involving equations over arbitrary languages. Nevertheless, we recall some of the related work on checking equivalence of regular expressions, for which a wide range of different tools and techniques has been developed. We only recall the ones most relevant to our work. CIRC [LGCR09] 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 [KN12] and formalized in type theory, yielding an implementation in Coq [CS11] (while [CS11] does not mention bisimulations explicitly, their method is based on constructing a bisimulation). There is another Coq implementation of regular expression equivalence, which is based on partial derivatives [MPdS12]. 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 [BP12]. We refer to [NT14] for an overview and comparison between these approaches. 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 chapter. In [Gra05], a proof system for equivalence of regular expressions is presented, based on bisimulations but not on bisimulation up-to. In [HN11], a general coinductive axiomatization of regular expression containment is given, based on an interpretation of regular expressions as types. The authors of [HN11] instantiate their axiomatization with the main coinductive rule from [Gra05]. The focus of [HN11] is on constructive proofs based on parse trees of regular expressions. In contrast, our approach is based on bisimulations between languages.

The presented proof techniques apply to undecidable problems such as language equivalence of context-free grammars. Indeed, automation is not aimed at in this chapter. Nevertheless, the present techniques can be seen as a foundation for novel interactive theorem provers, and extensions of fully automated tools such as [KN12, LGCR09, CS11].

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 relates 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 the subsequent chapters of this thesis for details. Bisimulation up to bisimilarity (alone, without context and equivalence closure) was originally introduced in [Mil83], and in the context of automata and languages *simulation up to similarity* was introduced in [Rut98a].