Generic Partition Refinement Algorithms for Coalgebras and an Instantiation to Weighted Automata

Barbara König and Sebastian Küpper

Universität Duisburg-Essen, Germany

Abstract. Coalgebra offers a general framework for modelling different types of state-based systems. Our aim is to present generic algorithms to decide behavioural equivalence for coalgebras which generalize partition refinement. The underlying idea of the algorithms is to work on the final chain and to factor out redundant information. If the algorithm terminates, the result of the construction is a representative of the given coalgebra that is not necessarily unique and that allows to precisely answer questions about behavioural equivalence. We apply the algorithm to weighted automata over semirings in order to obtain a procedure for checking language equivalence for a large number of semirings.

1 Introduction

Coalgebra [Rut00] offers a unifying theory in which we can model and reason about various types of transition systems and automata. Given a category $\mathbf{C}$ and an endofunctor $F\colon \mathbf{C} \to \mathbf{C}$, an F-coalgebra is an arrow $\alpha\colon X \to FX$. In the case of $\mathbf{Set}$, X can be seen as the set of states of the transition system, F represents the branching type and α is the transition function. Depending on the choice of X we can specify different kinds of systems: for instance, the (finite) powerset functor $FX = \mathcal{P}_{fin}(X)$ specifies non-deterministic branching, whereas the functor $FX = X^A$ describes labels. Such functors can be combined to construct more complex types of transition systems.

Coalgebra comes equipped with a canonical notion of behavioural equivalence [Sta09]. We believe that an important contribution of coalgebra should be the provision of generic algorithms for checking behavioural equivalence, independent of the type of transition system. Such generic algorithms would be useful for two reasons: for classifying and comparing existing algorithms and for obtaining prototype algorithms (that might be further optimized) when studying a new class of systems. One example of such generic methods that have recently been studied are up-to techniques [RBB$^+$14,BPPR14].

Here we are interested in generic algorithms for checking behavioural equivalence, akin to minimization or partition refinement techniques. A rather general account for such minimization techniques has been presented in [ABH$^+$12], by using factorization structures and factoring the arrows into the final chain. For coalgebras over $\mathbf{Set}$ this boils down to classical partition refinement, encompassing minimization of deterministic automata [HU79] or the computation of bisimulation equivalence classes for probabilistic transition systems [LS89,Bai96]. In [ABH$^+$12] we have also shown how to handle some coalgebras in categories different from $\mathbf{Set}$, especially in Kleisli categories, which allow

to specify side-effects and hence trace equivalence (where non-determinism is abstracted away as a side-effect).

However, some examples do not fall into this framework, most notably weighted automata over semirings, for which the underlying category does not possess suitable factorization structures. We found that a different notion of factorization can be used to capture behavioural equivalence. Furthermore it is unnecessary to look for a unique minimization or canonical representative, rather it is sufficient to compute *some* representative coalgebra and use it to precisely answer questions about behavioural equivalence. For weighted automata over semirings this yields an algorithm that we did not find in the literature as such. For probabilistic automata there is a related procedure for checking language equivalence [KMO$^+$11], for fields a method is discussed in [Bor09] and a result for rings, based on results by Schützenberger, is given in [DK13]. The notion of coalgebra homomorphism is related to conjugacy described in [BLS06].

We will present a generic algorithm, based on the notion of equivalence classes of arrows. We will compare this algorithm to the algorithm of [ABH$^+$12] that uses factorization structures [AHS90]. An important special case that we will discuss in detail and with many examples is the setting of weighted automata.

2 Preliminaries

We presuppose basic knowledge of category theory. We assume the reader is familiar with the notions of categories and functors. We are considering coalgebras in categories possibly different from $\mathbf{Set}$. However, in order to be able to speak about behavioural equivalence of states, we need to restrict to concrete categories, i.e. categories $\mathbf{C}$ with a faithful functor $U\colon \mathbf{C} \to \mathbf{Set}$, called concretization functor.

A coalgebra is an arrow $\alpha\colon X \to FX$ where the endofunctor F describes the branching type of the system under consideration, X plays the role of the set of states and α specifies transitions.

Definition 2.1 (Coalgebra). *Let $F : \mathbf{C} \to \mathbf{C}$ be an endofunctor on a category $\mathbf{C}$. An F-coalgebra is a pair $(X, \alpha : X \to FX)$, where X is an object of $\mathbf{C}$ and α is an arrow in $\mathbf{C}$. Given two F-coalgebras (X, α), (Y, β), a coalgebra homomorphism from (X, α) to (Y, β) is a $\mathbf{C}$-arrow $f : X \to Y$, so that $Ff \circ \alpha = \beta \circ f$.*

Coalgebra homomorphisms can be considered as structure-preserving maps between coalgebras, they correspond to functional bisimulations.

Definition 2.2 (Behavioural equivalence). *Let $(\mathbf{C}, U)$ be a concrete category and $(X, \alpha : X \to FX)$ be an F-coalgebra. We call the elements $x \in UX$ the states of (X, α). Two states $x, y \in UX$ are* behaviourally equivalent *($x \sim y$) if and only if there exists an F-coalgebra (Y, β) and a coalgebra-homomorphism $f : (X, \alpha) \to (Y, \beta)$ such that $Uf(x) = Uf(y)$.*

We will now introduce some relations on objects and arrows.

Definition 2.3 (Relations on objects and arrows). *Let X, Y be two objects of a category $\mathbf{C}$. We write $X \leq Y$ whenever there is an arrow $f\colon X \to Y$. Furthermore we write $X \equiv Y$ whenever $X \leq Y$, $Y \leq X$.*

Let $a\colon X \to A$, $b\colon X \to B$ be two arrows in $\mathbf{C}$ with the same domain. We write $a \leq^X b$ whenever there exists an arrow $d\colon A \to B$ with $d \circ a = b$. Similarly we write $a \equiv^X b$.

If the objects of a category formed a set, $\leq$ would be a preorder (or quasi-order) and $\equiv$ would be an equivalence relation. That is, reflexivity, transitivity and symmetry hold (for $\equiv$). Note that if a category has a final object 1, then $X \leq 1$ holds for any other object X. Furthermore if $f\colon X \to 1$ is an arrow into the final object, we have that $g \leq^X f$ for any other arrow $g\colon X \to Y$.

Proposition 2.4. *Let X, Y be objects of a category $\mathbf{C}$ and let $a\colon X \to A$, $b\colon X \to B$ be arrows in $\mathbf{C}$. Furthermore let $F\colon \mathbf{C} \to \mathbf{C}$ be an endofunctor. Then (i) $X \leq Y$ implies $FX \leq FY$, (ii) $a \leq^X b$ implies $Fa \leq^{FX} Fb$, (iii) $a \leq^X b$ implies $a \circ c \leq^Y b \circ c$ for any arrow $c\colon Y \to X$.*

Example 2.5. Let $f\colon X \to A$, $g\colon X \to B$ where $X \neq \emptyset$ be two functions in $\mathbf{Set}$. It holds that $f \equiv^X g$ if and only if both functions induce the same partition on X, i.e., for all $x, y \in X$ it holds that $f(x) = f(y) \iff g(x) = g(y)$. Similarly $f \leq^X g$ means for all $x, y \in X$ that $f(x) = f(y) \Rightarrow g(x) = g(y)$.

Hence if $a \equiv^X b$ holds for two arrows $a\colon X \to A$, $b\colon X \to B$ in a concrete category $(\mathbf{C}, U)$, we can conclude that Ua, Ub induce the same partition on UX.

The notion of equivalent arrows is connected to the notion of split-mono. An arrow $m\colon X \to Y$ is called split-mono if it has a left inverse, i.e., if there exists an arrow $m^{\leftarrow}\colon Y \to X$ such that $m^{\leftarrow} \circ m = id_X$. Now, assume two arrows $a\colon X \to Y$, $b\colon X \to Z$ are equivalent ($a \equiv^X b$). Then there is an arrow $m\colon Y \to Z$ and an arrow $m^{\leftarrow}\colon Z \to Y$ such that $m \circ a = b$ and $m^{\leftarrow} \circ b = a$. Then, $m^{\leftarrow} \circ m \circ a = a$, hence m behaves like a split-mono, relative to a. More concretely, split-monos $m\colon X \to Y$ are exactly the arrows that are equivalent to id_X.

In Section 4 we will show that equivalence on arrows boils down to a very natural notion in the setting of weighted automata: the fact that two sets of vectors (with entries from a semiring) generate the same semimodule.

Finally, we need the notion of factorization structures [AHS90].

Definition 2.6 (Factorization Structures). *Let $\mathbf{C}$ be a category and let $\mathcal{E}$, $\mathcal{M}$ be classes of morphisms in $\mathbf{C}$. The pair $(\mathcal{E}, \mathcal{M})$ is called a* factorization structure *for $\mathbf{C}$ whenever*

- *$\mathcal{E}$ and $\mathcal{M}$ are closed under composition with isos.*
- *$\mathbf{C}$ has* $(\mathcal{E}, \mathcal{M})$-factorizations *of morphisms, i.e., each morphism f of $\mathbf{C}$ has a factorization $f = m \circ e$ with $e \in \mathcal{E}$ and $m \in \mathcal{M}$.*
- *$\mathbf{C}$ has the unique* $(\mathcal{E}, \mathcal{M})$-diagonalization property*: for each commutative square $g \circ e = m \circ f$ with $e \in \mathcal{E}$ and $m \in \mathcal{M}$ there exists a unique* diagonal*, i.e., a morphism d such that $d \circ e = f$ and $m \circ d = g$.*

3 Generic Algorithms

For our algorithms we assume in the following that the category under consideration has a final object 1. Before introducing the algorithms, based on the construction of the final

chain [AK95], we will first discuss how behavioural equivalence can be expressed as a post-fixpoint using the terminology of the previous section.

Proposition 3.1. *Let F be an endofunctor on a concrete category $(\mathbf{C}, U)$ and let $\alpha : X \to FX$ be a coalgebra on $\mathbf{C}$. Furthermore let $f\colon X \to Y$ be an arrow. It holds that $f \leq^X Ff \circ \alpha$ (f is a post-fixpoint) if and only if there exists a coalgebra $\beta\colon Y \to FY$ such that f is a coalgebra homomorphism from (X, α) to (Y, β).*

For every such post-fixpoint f we have that $x, y \in UX$ and $Uf(x) = Uf(y)$ implies $x \sim y$. If, in addition, it holds for every other post-fixpoint $g\colon X \to Z$ that $g \leq^X f$ (f is the largest post-fixpoint), we can conclude that f induces behavioural equivalence, i.e., $Uf(x) = Uf(y) \iff x \sim y$.

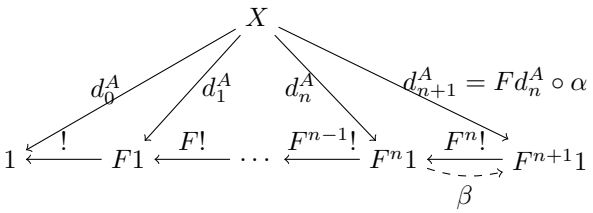
Proof. The first statement is almost trivial, since $f \leq^X Ff \circ \alpha$ means the existence of an arrow $\beta\colon X \to FX$ with $\beta \circ f = Ff \circ \alpha$, which is exactly the condition that β is a coalgebra homomorphism. Hence, by definition of behavioural equivalence, $Uf(x) = Uf(y)$ implies $x \sim y$. It is left to show that $x \sim y$ implies $Uf(x) = Uf(y)$ if f is the largest fixpoint. Since $x \sim y$, there must be some coalgebra $\gamma\colon Z \to FZ$ and a coalgebra homomorphism $g\colon X \to Z$ such that $\gamma \circ g = Fg \circ \alpha$ and $Ug(x) = Ug(y)$. This implies that $g \leq^X Fg \circ \alpha$ and hence $g \leq^X f$. Finally, we obtain $Uf(x) = Uf(y)$. □

In $\mathbf{Set}$ one can imagine the largest fixpoint $f\colon X \to Y$ as a function that maps every state into its equivalence class. We will now discuss three algorithms that expect a coalgebra α as input and yield, if they terminate, a coalgebra homomorphism (from α to some target coalgebra) that induces behavioural equivalence.

Algorithm 3.2 (Final Chain Algorithm A) *Let F be an endofunctor on a concrete category $(\mathbf{C}, U)$ and let $\alpha : X \to FX$ be a coalgebra in $\mathbf{C}$. We define the following algorithm.*

Step 0**:** *Take the (unique) arrow $d_0^A\colon X \to 1$.*
Step $i+1$**:** *Compute $d_{i+1}^A = Fd_i^A \circ \alpha : X \to F^{i+1}1$.*
If there exists an arrow $\beta : F^i1 \to F^{i+1}1$ such that $\beta \circ d_i^A = d_{i+1}^A$, i.e., if $d_i^A \leq^X d_{i+1}^A$, the algorithm terminates and returns d_i^A and (F^i1, β) as its result.

Algorithm A is well-known iteration along the final chain, however, to our knowledge, the termination condition is new.



The algorithm is easy to analyse using the terminology introduced earlier. Specifically, it yields a sequence of arrows $d_0^A \geq^X d_1^A \geq^X d_2^A \geq^X \dots$ that approximates behavioural equivalence from above. It terminates whenever this sequence becomes stationary, i.e., $d_n^A \equiv^X d_{n+1}^A$.

Lemma 3.3. *Let $g : X \to Z$ be any post-fixpoint, i.e. $g \leq^X Fg \circ \alpha$, then for all d_i^A obtained in Algorithm A we have $d_i^A \geq^X g$.*

Proof. Clearly $d_0^A \geq^X g$, because d_0^A is the arrow into the final object of the category. By induction, using Proposition 2.4, we can show that $d_i^A \geq^X g$ implies $d_{i+1}^A = Fd_i^A \circ \alpha \geq^X Fg \circ \alpha \geq^X g$. □

Proposition 3.4. *If Algorithm A terminates in step n, its result d_n^A induces behavioural equivalence, i.e. $x, y \in UX$ are behaviourally equivalent ($x \sim y$) if and only if $Ud_n^A(x) = Ud_n^A(y)$.*

Proof. First, $d_0^A \geq^X d_1^A$ since d_0^A is an arrow into the final object. Since $\leq^X$ is preserved by functors and by right composition with arrows, we obtain by induction, using Proposition 2.4, that $d_i^A \geq^X d_{i+1}^A$. The termination condition says that $d_n^A \leq^X d_{n+1}^A = Fd_n^A \circ \alpha$. In order to conclude with Proposition 3.1 that the resulting arrow d_n^A induces behavioural equivalence, we have to show that it is the largest post-fixpoint. Assume another post-fixpoint $g\colon X \to Z$ with $g \leq^X Fg \circ \alpha$. Then, Lemma 3.3 shows that $d_n^A \geq^X g$ and thus d_n^A is the largest post-fixpoint. □

Furthermore whenever $d_n^A \equiv^X d_{n+1}^A$ we have that $d_n^A \equiv^X d_m^A$ for every $m \geq n$. Hence every arrow d_m^A obtained at a later step induces behavioural equivalence as well. We can show that d_n^A is equivalent to the arrow into the final coalgebra (if it exists).

Lemma 3.5. *Let (Z, κ) be the final coalgebra, i.e., a coalgebra into which there exists a unique coalgebra homomorphism from every other coalgebra. Take a coalgebra (X, α) and the unique homomorphism $f\colon (X, \alpha) \to (Z, \kappa)$ and assume that Algorithm A terminates in step n. Then $f \equiv^X d_n^A$.*

Note that after each step of the algorithm, it is permissible to replace d_i^A with any representative e_i^A of the equivalence class of d_i^A, i.e., any e_i^A with $d_i^A \equiv^X e_i^A$. This holds because $d_i^A \equiv^X e_i^A$ implies $d_{i+1}^A = Fd_i^A \circ \alpha \equiv^X Fe_i^A \circ \alpha$ and checking the termination condition $d_n^A \leq^X d_{n+1}^A$ can instead be done for any representatives of d_n^A, d_{n+1}^A. This gives rise to the following algorithm:

Algorithm 3.6 (Final Chain Algorithm B) *Let F be an endofunctor on a concrete category $(\mathbf{C}, U)$ and let $\alpha : X \to FX$ be a coalgebra in $\mathbf{C}$. Morover, let $\mathcal{R}$, the class of representatives, be a class of arrows of $\mathbf{C}$ such that for any arrow d in $\mathbf{C}$ we have an arrow $e \in \mathcal{R}$ that is equivalent to d, i.e. $d \equiv^X e$. We define the following algorithm:*

Step 0**:** *Take the (unique) arrow $d_0^B\colon X \to 1$.*
Step $i+1$**:** *Find a representative $e_i^B \in \mathcal{R}$, for d_i^B, i.e. factor $d_i^B = m_i^B \circ e_i^B$ such that $(e_i^B\colon X \to Y_i) \in \mathcal{R}$, $m_i^B\colon Y_i \to FY_{i-1}$. Determine $d_{i+1}^B = Fe_i^B \circ \alpha\colon X \to FY_i$. If there exists an arrow $\gamma\colon Y_i \to FY_i$ such that $\gamma \circ e_i^B = d_{i+1}^B$, i.e., $d_{i+1}^B \geq^X e_i^B$, the algorithm terminates and returns e_i^B and (Y_i, γ) as its result.*

Choosing good and compact representatives can substantially mitigate state space explosion. Hence, Algorithm B is an optimization of Algorithm A that (potentially) reduces the number of computations needed in every step. Moreover, Algorithm B terminates in exactly as many steps as Algorithm A (if it terminates).

Proposition 3.7. *Algorithm B terminates in n steps if and only if Algorithm A terminates in n steps. Furthermore two states $x, y \in UX$ are behaviourally equivalent ($x \sim y$) if and only if $Ue_n^B(x) = Ue_n^B(y)$.*

Termination for Algorithm B is independent of the choice of the representatives e_i^B.

In [ABH+12] an algorithm similar to ours is being discussed. The algorithm also works on the final chain and uses a factorization system to trim the intermediate results with the aim of finding a so-called minimization, i.e., a unique and minimal representative of the given coalgebra α. We we will discuss a variation of said algorithm.

Algorithm 3.8 (Final Chain Algorithm C) *Let F be an endofunctor on a concrete category $(\mathbf{C}, U)$ and let $\alpha : X \to FX$ be a coalgebra in $\mathbf{C}$. Morover, let $(\mathcal{E}, \mathcal{M})$ be a factorization structure such that U maps $\mathcal{M}$-morphisms to injections. We define the following algorithm:*

Step 0**:** *Take the (unique) arrow $d_0^C \colon X \to 1$.*
Step $i+1$**:** *Factor $d_i^C = m_i^C \circ e_i^C$ via the factorization structure, i.e. $(e_i^C \colon X \to Y_i) \in \mathcal{E}$, $(m_i^C \colon Y_i \to FY_{i-1}) \in \mathcal{M}$. Determine $d_{i+1}^C = Fe_i^C \circ \alpha \colon X \to FY_i$.*
If there exists an arrow $\delta \colon Y_i \to FY_i$ such that $\delta \circ e_i^C = d_{i+1}^C$, the algorithm terminates and returns e_i^C and (Y_i, δ) as its result.

Remark 3.9. Note that for Algorithms A and B we have to assume that the preorders are decidable, for Algorithms B and C we have to assume that the factorizations are computable. Naturally, in concrete applications, one has to choose a suitable class of representatives and a strategy for factoring. For the case of weighted automata we will discuss how such a strategy looks like (see Section 4).

If a category has a suitable factorization structure that ensures compact intermediate results, Algorithm C might terminate faster than Algorithm B. However, not every category has a suitable factorization structure (see our examples in Section 4), which drove us to investigate alternatives such as Algorithms A and B introduced earlier. For the trivial factorization structure where $\mathcal{E}$ is the class of all arrows and $\mathcal{M}$ are just the isomorphisms, we obtain the unoptimized Algorithm A. Algorithm C is a variation of the algorithm in [ABH+12] with a relaxed (but correct) termination condition: in the earlier version the algorithm terminated only if one of the m_i^C was an isomorphism.

Under certain conditions, stated below, Algorithm C terminates in the same number of steps as Algorithm B.

Proposition 3.10. *Let F be an endofunctor on a concrete category $(\mathbf{C}, U)$ and let $\alpha : X \to FX$ be a coalgebra in $\mathbf{C}$. Moreover, let $(\mathcal{E}, \mathcal{M})$ be a factorization structure for $\mathbf{C}$.*

1. *Assume that F preserves $\mathcal{M}$-arrows. If Algorithm A terminates in n steps, then Algorithm C terminates in n steps as well.*
2. *If Algorithm C terminates in n steps and for each arrow d_i^A, $i = 0, \ldots, n$ there exists an arrow $e_i \in \mathcal{E}$ with $e_i \equiv^X d_i^A$ (i.e., there is an arrow from $\mathcal{E}$ in all relevant equivalence classes), then Algorithm B terminates in n steps as well.*

Furthermore two states $x, y \in UX$ are behaviourally equivalent ($x \sim y$) if and only if $Ue_n^C(x) = Ue_n^C(y)$.

In $\mathbf{Set}$ every equivalence class of arrows, except the equivalence classes of $\equiv^{\emptyset}$, contains a surjection. Hence if we choose surjections and injections as a factorization structure in $\mathbf{Set}$, we obtain exactly the same termination behaviour as the other two algorithms, provided that the state set is not empty. In this case, a suitable class $\mathcal{R}$ of representatives for Algorithm B would also be the surjections. This yields classical partition refinement algorithms.

Termination: The algorithm terminates whenever $\equiv^X$ has only finitely many equivalence classes. In $\mathbf{Set}$ this is the case for finite sets X. However, there are other categories where this condition holds for certain objects X: in $\mathbf{Vect}$, the category of vector spaces and linear maps, $\equiv^X$ is of finite index, whenever X is a finite-dimensional vector space.

Note that it is not sufficient to iterate until the partition induced by Ud_i is not refined from one step to the next. We will discuss weighted automata in the next section and the tropical semiring provides an example where behavioural equivalence is undecidable, but all steps of Algorithm B are computable and the partition induced by Ud_i can only be refined finitely often, rendering this criterion invalid.

4 Weighted Automata

We will show that we can apply our approach to weighted automata with weights taken from a semiring. Weighted automata [DKV09] are a versatile tool to specify and analyse systems equipped with quantitative information. They are a generalization of non-deterministic automata (NFA), but instead of just accepting or rejecting words over a given alphabet, weighted automata assign values taken from a semiring to words.

Coalgebraic treatment for weighted automata was already discussed in [BMS13], but for different categories. Here, we follow the ideas of [HJS07], which shows how to obtain trace or language equivalence by working in a Kleisli category.

We mainly consider Algorithm B. The algorithm will not terminate for every possible choice of the semiring (for example it is known that language equivalence is undecidable for tropical semirings). However, we will characterize the cases for which it terminates.

Definition 4.1 (Semiring). *Let S be a set. A* semiring *is a tuple $\mathbb{S} = (S, +, \cdot, 0, 1)$, where $0 \in S$, $1 \in S$ and $1 \neq 0$, $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, $0 \cdot a = a \cdot 0 = 0$ for all $a \in S$ and the distributive laws $(a + b) \cdot c = a \cdot c + b \cdot c$ and $c \cdot (a + b) = c \cdot a + c \cdot b$ hold for all $a, b, c \in S$. We will in the sequel identify $\mathbb{S}$ with the set S it is defined on.*

Hence a semiring is a ring that need not have additive inverses. All rings and therefore all fields are semirings. Whenever we are talking about a semiring throughout this paper, we assume $\mathbb{S}$ is a decidable set and the operations $\oplus$ and $\otimes$ can be computed effectively.

Example 4.2. The semiring $\mathbb{S} = (\mathbb{N} \cup \{\infty\}, min, +, \infty, 0)$, where $\mathbb{S}$-addition is the minimum and $\mathbb{S}$-multiplication is addition, is called the tropical semiring.

Take a partially ordered set $(\mathbb{L}, \leq)$ such that for $a, b \in \mathbf{L}$ we have a supremum $a \sqcup b$ and an infimum $a \sqcap b$. If $\mathbf{L}$ is a distributive lattice and we have top and bottom elements $\top$, $\bot$, we obtain a semiring $(\mathbb{L}, \sqcup, \sqcap, \bot, \top)$.

The usual notions of linear algebra (vector spaces, matrices, linear maps) can be extended from fields to semirings.

Definition 4.3 (Semimodule, matrix). *Let X be an index set, let $\mathbb{S}$ be a semiring and let $\mathbb{S}^X$ the set of all functions $s\colon X \to \mathbb{S}$. Then $\mathbb{S}^X$ is a* semimodule*, i.e., a set closed under pointwise addition and multiplication with a constant. Every subset of $\mathbb{S}^X$ closed under these operations is called a* subsemimodule *of $\mathbb{S}^X$.*

An $X \times Y$-matrix a with weights over $\mathbb{S}$ is given by a function $a\colon X \times Y \to \mathbb{S}$, i.e., it is an element of $\mathbb{S}^{X \times Y}$, each row $x \in X$ there are only finitely many entries different from 0. We can multiply an $X \times Y$-matrix a and a $Y \times Z$-matrix b in the usual way, obtaining an $X \times Z$-matrix $a \cdot b$.

By $a_x\colon Y \to \mathbb{S}$, where $x \in X$, we denote the i-th row of a, i.e., $a_x(y) = a(x, y)$.

Similarly, for a generating set $G \subseteq \mathbb{S}^Y$ of vectors we denote by $\langle G \rangle$ the subsemimodule spanned by G, i.e., the set that contains all linear combinations of vectors of G. Given an $X \times Y$-matrix we denote by $\langle a \rangle \subseteq \mathbb{S}^Y$ the subsemimodule of $\mathbb{S}^Y$ that is spanned by the rows of a, i.e., $\langle a \rangle = \langle \{a_x \mid x \in X\} \rangle$.

Since matrices are arrows in the category, we represent them by lower case letters.

Definition 4.4 (Category of matrices, linear maps). *We consider a category $\mathbf{M}(\mathbb{S})$ of $\mathbb{S}$-matrices where objects are sets and an arrow $a\colon Y \to X$ is an $X \times Y$-matrix a as defined above. Arrow composition is performed by matrix multiplication, i.e., for $a\colon X \to Y$, $b\colon Y \to Z$ we have $b \circ a = b \cdot a$. The identity arrow $id_X\colon X \to X$ is the $X \times X$ unit matrix.*

There exists a concretization functor U where $UX = \mathbb{S}^X$ and for $a\colon X \to Y$ $Ua : UX \to UY$ is the linear map from $\mathbb{S}^X$ to $\mathbb{S}^Y$ represented by the matrix a, i.e., $Ua(x) = a \cdot x$ for all $x \in \mathbb{S}^X$. (Note that x is considered as a vector and the multiplication of a matrix with a vector is defined as usual.)

If the semiring is a field, there exists a factorization structure which factors every matrix into a matrix of full row rank and a matrix of full column rank. This factorization is unique up-to isomorphism. However, for generic semirings, semimodule theory does not provide a similarly elegant notion of basis as in vector spaces, and such unique factorizations are not possible in general. Hence Algorithm C is usually not applicable.

We can now define weighted automata in a coalgebraic notation. Note that, different from the weighted automata in [DKV09], our automata do not have initial states, as it is customary in coalgebra, and hence no initial weights, but only final weights. However, for language equivalence we can easily simulate initial weights by adding a new state to the automaton with edges going to each state of the automaton, carrying the initial weights of these states and labelled with some (new) symbol.

Definition 4.5 (Weighted Automaton). *Let $\mathbf{M}(\mathbb{S})$ be the category defined above. Let A be a finite set of alphabet symbols. We define an endofunctor $F\colon \mathbf{M}(\mathbb{S}) \to \mathbf{M}(\mathbb{S})$ as*

follows: on objects[1] $FX = A \times X + 1$ *for a set* X*. For an arrow* $f\colon Y \to X$ *we have* $Ff\colon A \times Y + 1 \to A \times X + 1$ *where* $Ff((a,x),(a,y)) = f(x,y)$ *for* $a \in A$*,* $x \in X$*,* $y \in Y$*,* $Ff(\bullet,\bullet) = 1$ *and* $Ff(c,d) = 0$ *for all remaining* $c \in A \times X + 1, d \in A \times Y + 1$*.*

A weighted automaton *is an* F*-coalgebra, i.e., an arrow* $\alpha\colon X \to FX$ *in the category* $\mathbf{M}(\mathbb{S})$ *or, alternatively, an* $FX \times X$*-matrix with entries taken from* $\mathbb{S}$*.*

For a weighted automaton α, $\alpha(\bullet, x)$ denotes the final weight of state $x \in X$ and $\alpha((a,y),x)$ denotes the weight of the a-transition from x to y. We are mainly interested in weighted automata where the state set X is finite.

Definition 4.6 (Language of a Weighted Automaton). *Let* (X,α) *be a weighted automaton over alphabet* A*, a semiring* $\mathbb{S}$ *and a finite state set* X*. The language* $L_\alpha : A^* \to \mathbb{S}^X$ *of* α *is recursively defined as*

- $L_\alpha(\varepsilon)(x) = \alpha(\bullet, x)$
- $L_\alpha(aw)(x) = \sum_{x' \in X} \alpha((a,x'),x) \cdot L_\alpha(w)(x')$ *for* $a \in A$*,* $w \in A^*$

We will call $L_\alpha(w)(x)$ *the weight that state* x *assigns to the word* w*.*

Remark 4.7. Note that the language of a weighted automaton need not be defined if X is not finite, because coalgebras in $\mathbf{M}(\mathbb{S})$ need not be finitely branching and therefore computing the language of a state may depend on computing an infinite sum.

It can be shown that two states $x, y \in X$ of a finite weighted automaton (X,α) are behaviourally equivalent in the coalgebraic sense ($x \sim y$) if and only if they assign the same weight to all words, i.e. $L_\alpha(w)(x) = L_\alpha(w)(y)$ for all $w \in A^*$.

Proposition 4.8. *Let* $(X, \alpha\colon X \to FX)$*, where* X *is a finite set, be a weighted automaton over the finite alphabet* A *and the semiring* $\mathbb{S}$*. Then, two states* $x, y \in X$ *are language equivalent if and only if they are behaviourally equivalent.*

Due to the change of category (from $\mathbf{Set}$ to $\mathbf{M}(\mathbb{S})$) we obtain language equivalence instead of a notion of bisimilarity. Note that $\mathbf{M}(\mathbb{S})$ could also be considered as a Kleisli category over the semiring monad (see also [HJS07] which explains the effects that the implicit branching or side-effects of the monad have on behavioural equivalence).

From now onwards we consider only finite index sets. We will study the category $\mathbf{M}(\mathbb{S})$ and show what the preorder on arrows (see Definition 2.3) means in this setting.

Proposition 4.9. *Let* $a\colon X \to Y$*,* $b\colon X \to Z$ *be two arrows in* $\mathbf{M}(\mathbb{S})$*, i.e.,* a *is a* $Y \times X$*-matrix and* b *is a* $Z \times X$*-matrix. It holds that* $a \leq^X b \iff \langle a \rangle \subseteq \langle b \rangle$*. That is, two matrices* a, b *are ordered if and only if the subsemimodule spanned by* a *is included in the subsemimodule spanned by* b*. Hence also* $a \equiv^X b \iff \langle a \rangle = \langle b \rangle$*.*

We will now describe how to choose representatives in this category. Given an arrow $a\colon X \to Y$, i.e., a $Y \times X$-matrix, choose any set of vectors $G \subseteq \mathbb{S}^X$ that generates $\langle a \rangle$, i.e., $\langle a \rangle = \langle G \rangle$, and take any matrix a' with G as row vectors. Then $a \equiv^X a'$ and a' can be taken as a representative of a.

[1] Here $X + Y$ denotes the disjoint union of two sets X, Y and 1 stands for the singleton set $\{\bullet\}$.

Finding such a set G of generators that is minimal in size will be quite difficult for a general semiring, since semimodules do not have a notion of basis as elegant as vector spaces. However, for our optimization purposes, it is enough to make a' reasonably small. This can be done by going through the row vectors of a in any order, eliminating every vector that can be written as a linear combination of the remaining row vectors. We will use this strategy to determine representatives in $\mathcal{R}$ for Algorithm B, which are those matrices where no further eliminations are possible.

Definition 4.10 (Class of Representatives). *We define $\mathcal{R}$ as the class of all matrices a that do not contain a row that is a linear combination of the other rows of a.*

Our algorithm need not terminate though. Termination depends on the semiring and possibly on the automaton we investigate. It was already shown that language equivalence is not decidable for weighted automata with weights over the *tropical semiring* as shown in [Kro92,ABK11]. To state a termination condition, we define the following generating sets for a given weighted automaton (X, α): $S^n = \{L_\alpha(w) \mid w \in A^*, |w| \leq n\}$ and $S^* = \{L_\alpha(w) \mid w \in A^*\} = \bigcup_{n=0}^{\infty} S^n$. It can be shown that $S^* \subseteq \mathbb{S}^X$ is finitely generated if and only if there exists an index n such that $\langle S^n \rangle = \langle S^* \rangle$. (Note that $\langle S^* \rangle$ consists of all linear combinations of vectors of S^*. Hence if it has a finite generator set G, the vectors of G can again be composed of finitely many vectors of S^*. And these must be contained in some S^n.) We will show that Algorithm B terminates whenever $\langle S^* \rangle$ is finitely generated (see also [DK13,BMS13]). We start with the following lemma:

Lemma 4.11. *Let $d_i \colon X \to F^i 1$, $i \in \mathbb{N}_0$ be the sequence of arrows resp. matrices generated by the Algorithm A, where 1 (the final object) is the empty set. Hence $F^i 1$ contains tuples of the form $(a_1, a_2, \ldots, a_{i-1}, \bullet)$, where $a_j \in A$. We will identify such a tuple with the word $w = a_1 a_2 \ldots a_{i-1}$. Then, for an index $i \in \mathbb{N}_0$ and a word w with $|w| < i$, we have $(d_i)_w = L_\alpha(w)$. This means that $\langle d_{i+1} \rangle = \langle S^i \rangle$.*

Theorem 4.12. *If $\langle S^* \rangle$ for a weighted automaton (X, α) is finitely generated, Algorithm B terminates.*

Proof. From Lemma 4.11 it follows that $\langle d^A_{i+1} \rangle = \langle S^i \rangle$. Since $\langle S^* \rangle$ is finitely generated, there must be an index n with $\langle S^n \rangle = \langle S^{n+1} \rangle$, hence $\langle d^A_{n+1} \rangle = \langle d^A_{n+2} \rangle$. Proposition 4.9 then implies that $d^A_{n+1} \equiv^X d^A_{n+2}$ and hence $d^A_{n+1} \leq^X d^A_{n+2}$, which is exactly the termination condition. Note that this is not impaired when we consider representatives e_i in Algorithm B, since $e^B_{n+1} \leq^X e^B_{n+2}$ holds as well. In fact, a representative e^B_i generates the same subsemimodule as d^A_i. □

If $\mathbb{S}$ is a *field*, we basically obtain the algorithm for fields from [Bor09,ABH$^+$12]. Note that for fields, minimal generating sets of semimodules, i.e. bases for vector spaces, always have the same size, so each intermediate result will be of the smallest size possible. Using a result by Droste and Kuske, [DK13], we can see that the algorithm also terminates for skew-fields, so structures that are almost fields, but where multiplication need not be commutative. In both cases, for a finite set X there are only $|X|$ different non-isomorphic semimodules, i.e., vector spaces. Thus any increasing chain has to become stationary and we can guarantee termination.

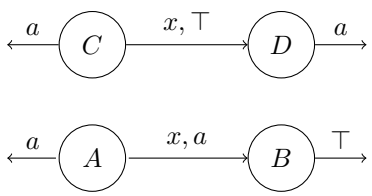
If $\mathbb{S}$ is a *finite semiring*, the algorithm terminates since there are only finitely many different row vectors of a fixed dimension $|X|$. If $\mathbb{S}$ is a *distributive complete lattice* or any other *commutative idempotent semiring*, the algorithm will terminate as well (because from the, obviously, finitely many weights contained in a finite automaton, only finitely many elements of $\mathbb{S}$ can be obtained via meet and join and thus there are again only finitely many different row vectors of a fixed dimension).

Note that termination does not necessarily imply decidability of language equivalence. For this it is also necessary to be able to decide whether two matrices are equivalent, i.e., whether a vector is generated by a given set of generators. This need not be decidable in general, but it is decidable in all the cases above.

We now consider several examples.

Example 4.13. We use as a semiring the complete distributive lattice $\mathbb{L} = \{\top, \bot, a, b\}$ where $\bot \leq a \leq \top$, $\bot \leq b \leq \top$ and consider the labelled transition system (X, α), $X = \{A, B, C, D\}$ with weights over $\mathbb{L}$ and labels from $A = \{x\}$ represented by the transition matrix and automaton (transitions with weight $\bot$ are omitted):

$$\alpha = \begin{array}{c} \\ (x,A) \\ (x,B) \\ (x,C) \\ (x,D) \\ \bullet \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} \bot & \bot & \bot & \bot \\ a & \bot & \bot & \bot \\ \bot & \bot & \bot & \bot \\ \bot & \bot & \top & \bot \\ a & \top & a & a \end{pmatrix} \end{array}$$



We apply Algorithm B. Below e_i denotes the chosen representative in $\mathcal{R}$.

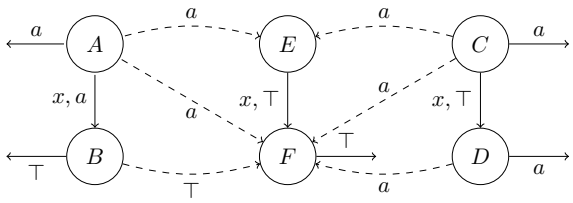
- We start with $d_0 = e_0$, a 0×4-matrix
- $Fe_0 = \begin{pmatrix} \bot & \bot & \bot & \bot & \top \end{pmatrix}$, $d_1 = Fe_0 \cdot \alpha = \begin{pmatrix} a & \top & a & a \end{pmatrix}$. This is in $\mathcal{R}$, so $e_1 = d_1$.
- $Fe_1 = \begin{pmatrix} a & \top & a & a & \bot \\ \bot & \bot & \bot & \bot & \top \end{pmatrix}$, $d_2 = Fe_1 \cdot \alpha = \begin{pmatrix} a & \bot & a & \bot \\ a & \top & a & a \end{pmatrix}$. This is in $\mathcal{R}$, so $e_2 = d_2$.
- $Fe_2 = \begin{pmatrix} a & \bot & a & \bot & \bot \\ a & \top & a & a & \bot \\ \bot & \bot & \bot & \bot & \top \end{pmatrix}$, $d_3 = Fe_2 \cdot \alpha = \begin{pmatrix} \bot & \bot & \bot & \bot \\ a & \bot & a & \bot \\ a & \top & a & a \end{pmatrix}$.

  This is not in $\mathcal{R}$, we can for example see that the first row equals $\bot$ times the second plus $\bot$ times the third row. So we factorize:

$$d_3 = \gamma \cdot e_3 = \begin{pmatrix} \bot & \bot \\ \top & \bot \\ \bot & \top \end{pmatrix} \cdot \begin{pmatrix} a & \bot & a & \bot \\ a & \top & a & a \end{pmatrix}$$

  Now $e_3 = e_2$ and we can stop our computation.

The resulting automaton is a two-state automaton, where the states will be called E, F. Looking at e_3 we can see that the states A and C of X are equivalent, since their columns coincide (in fact, both accept ε and x with weight a). States B and D on the other hand are not equivalent to any other state. We see that $e_2 : X \to Y$, where $Y = \{E, F\}$, is a coalgebra homomorphism from (X, α) to (Y, γ). The coalgebra (Y, γ) can be considered as a minimal representative of (X, α). The following diagram depicts automata (X, α) and (Y, γ) where the coalgebra homomorphism e_2 is drawn with dashed lines.



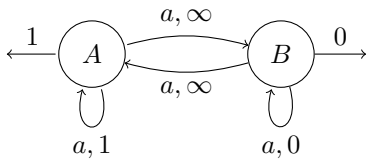
$$\gamma = \begin{matrix} \\ (x,E) \\ (x,F) \\ \bullet \end{matrix} \begin{matrix} E & F \end{matrix} \begin{pmatrix} \bot & \bot \\ \top & \bot \\ \bot & \top \end{pmatrix}$$

We can also see that our method of factoring is not unique, because in γ we could have chosen $\gamma((x,E),E) = b$ (and all other entries as before). In this case, there would be an x,b-loop on state E in the diagram above. Since all dashed arrows going into E carry weight a and $a \sqcap b = \bot$, this loop would not have any effect and the equivalence one obtains is the same.

In the next example, we will investigate the tropical semiring (cf. Example 4.2). We will write $\oplus$ and $\otimes$ for the $\mathbb{S}$-addition respectively the $\mathbb{S}$-multiplication to avoid confusion. Language equivalence is in general undecidable, hence the algorithm will not terminate in general.

Example 4.14. Consider the (rather simple) transition system over the one-letter alphabet $\{a\}$, given by the matrix α:

$$\begin{matrix} \\ (a,A) \\ (a,B) \\ \bullet \end{matrix} \begin{matrix} A & B \end{matrix} \begin{pmatrix} 1 & \infty \\ \infty & 0 \\ 1 & 0 \end{pmatrix}$$



Applying Algorithm B to α, we obtain the following (intermediate) results:

- $d_0 = e_0$ is the 0×2-matrix
- $Fe_0 = \begin{pmatrix} \infty & \infty & 0 \end{pmatrix}$, $d_1 = Fe_0 \cdot \alpha = \begin{pmatrix} 1 & 0 \end{pmatrix} = e_1$
- $d_2 = \begin{pmatrix} 2 & 0 \\ 1 & 0 \end{pmatrix} \in \mathcal{R}$, so we choose $e_2 = d_2$.
- $d_3 = \begin{pmatrix} 3 & 0 \\ 2 & 0 \\ 1 & 0 \end{pmatrix} \notin \mathcal{R}$, because we can obtain the second row as a linear combination of the first and the third row: $1 \otimes \begin{pmatrix} 1 & 0 \end{pmatrix} \oplus 0 \otimes \begin{pmatrix} 3 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \end{pmatrix} \oplus \begin{pmatrix} 3 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 0 \end{pmatrix}$
  However, we cannot obtain the first row via linear combination of the other two rows, so the algorithm cannot stop in the third iteration.
  So we can choose: $m_3 = \begin{pmatrix} 0 & \infty \\ 0 & 1 \\ \infty & 0 \end{pmatrix}$ and $e_3 = \begin{pmatrix} 3 & 0 \\ 1 & 0 \end{pmatrix}$
- From now on, each step is analogous to the third step, we obtain $e_i = \begin{pmatrix} i & 0 \\ 1 & 0 \end{pmatrix}$ in each iteration i, but we will never reach a d_{i+1} such that $d_{i+1} \equiv^X e_i$.

Algorithm B therefore does not terminate for α. However, since the two states are already separated from the first step onwards, we can at least conclude that they are not behaviourally equivalent. There are other example automata over this semiring for which Algorithm B does terminate.

5 Conclusion

Related Work: Our work is closely related to [ABH$^+$12] which uses factorization structures in order to obtain generic algorithms for partition refinement. However, the algorithm in [ABH$^+$12] could not handle general weighted automata over semirings, due to the absence of suitable factorization structures.

[Sta09] also discusses several coalgebraic methods to check behavioural equivalence, however the paper focusses more on the relation-based view with fixpoint iteration to obtain bisimulations. Staton compares with the final chain and can prove that whenever the arrow $F^i!$ in the final chain is a mono, then the relation refinement sequence converges at the latest. In our examples, the algorithm usually terminates earlier, since we only need a relative inverse β of $F^i!$ wrt. d_i^A.

For weighted automata, we are only aware of minimization algorithms for deterministic automata that do not work in the non-deterministic case. These algorithms are based on the idea of redistributing the weights of transitions in a canonical way and applying the minimization algorithm for deterministic automata (without weights) to the resulting automaton, where label and weight of each transition form a new transition label in the deterministic weight-free automaton. Mohri's algorithm [Moh97,Moh09] is based on weight pushing, and is applicable whenever $\mathbb{S}$ is zero-sum-free and weakly divisible. Eisner's algorithm [Eis03] works whenever $\mathbb{S}$ has multiplicative inverses. He remarks that his variation of weight pushing can also be applied to some semirings that do not have inverses, if they can be extended to ones that do have inverses. However, then, the minimal automaton might carry weights outside of $\mathbb{S}$.

Kiefer et al. [KMO$^+$11] have investigated optimizations for the case of weighted automata with weights over the field $\mathbb{R}$, with applications to probabilistic automata. Their algorithm is a probabilistic optimization of an algorithm that enumerates words of length at most n, where n is the number of states, and their weights. Instead of checking every such word they use probabilistic methods to determine which words to check. As far as we know this method can not easily be generalized to arbitrary semirings.

Language equivalence is not decidable for every semiring. For instance, it was shown that language equivalence is undecidable for the tropical semiring [Kro92,ABK11].

Droste and Kuske [DK13] describe when equivalence of two weighted automata is decidable, based on earlier results by Schützenberger. Their decidability result is close to our own, only they work with weighted automata with initial weights. They show that whenever $\mathbb{S}$ is a ring such that every subsemimodule generated by an automaton is finitely generated, language equivalence is decidable. While their notation, using linear presentations, is different from ours and they concentrate only on rings, the underlying ideas are related (cf. Section 4 where we show termination if the subsemimodule generated by an automaton is finitely generated).

In [BLS06], which uses results for rational power series presented in [BR88], it is shown that for subsemirings of a field (such as $\mathbb{Z}$ or $\mathbb{N}$) weighted language equivalence is decidable. The paper uses the notion of conjugacy, which is strongly related to the notion of coalgebra homomorphism.

Ésik and Maletti have investigated proper semirings in [ÉM10] and have proven that language equivalence for weighted automata is decidable whenever the corresponding semiring is proper and effectively presentable. Furthermore they investigated

Noetherian semirings, i.e. semirings where every subsemimodule of a finitely generated $\mathbb{S}$-semimodule is finitely generated. However, they do not give a concrete algorithm.

Bonsangue, Milius and Silva [BMS13] have also investigated language equivalence of weighted automata in a coalgebraic setting, however they use Eilenberg-Moore instead of Kleisli categories. They present an axiomatization of weighted automata and give a set of sound and complete equational laws axiomatizing language equivalence.

Recently, in [UH14], Urabe and Hasuo studied simulation and language inclusion for weighted automata, were the coalgebraic theory provides the basis for an efficient implementation.

We did not find our procedure for checking language equivalence for weighted automata in the literature, but it may not be too surprising, given the related work. However, we believe that our contribution goes beyond providing a procedure for weighted automata: we give a uniform presentation of generic algorithms that can be applied to various kinds of transition systems. Furthermore we attempted to bridge the gap between coalgebraic methods and methods for weighted automata.

Future Work: For future work we plan to further investigate the issue of termination: are there more general criteria which guarantee termination for our algorithms? For this we will need a suitable notion of "finiteness" in a general categorical setting, for instance the notion of *finitely generated* or *locally presentable* [AR94].

Furthermore, when working with equivalence classes of arrows, it is necessary to find good representatives, in order to discard redundant information and make the representation more compact.

So far we have studied coalgebras over $\mathbf{Set}$ and weighted automata in this setting. Furthermore our theory also applies to other examples in [ABH$^+$12] such as non-deterministic automata and conditional transition systems. Naturally, we plan to investigate additional case studies.

In [FMT05], a procedure similar to our algorithm is used to minimize HD-automata. We plan to check whether this algorithm is an instance of our Algorithm B.

Acknowledgements: We would like to thank Alexandra Silva, Filippo Bonchi and Marcello Bonsangue for several interesting discussions on this topic.

References

ABH$^+$12. J. Adámek, F. Bonchi, M. Hülsbusch, B. König, S. Milius, and A. Silva. A coalgebraic perspective on minimization and determinization. In *Proc. of FOSSACS '12*, pages 58–73. Springer, 2012. LNCS/ARCoSS 7213.

ABK11. S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *Proc. of ATVA*, volume 6996 of *LNCS*, pages 482–491. Springer, 2011.

AHS90. J. Adámek, H. Herrlich, and G.E. Strecker. *Abstract and Concrete Categories - The Joy of Cats*. Wiley, 1990.

AK95. J. Adámek and V. Koubek. On the greatest fixed point of a set functor. *Theoretical Computer Science*, 150:57–75, 1995.

AR94. J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories*, volume 189 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1994.

Bai96. C. Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In *Proc. of CAV '96*, pages 50–61. Springer, 1996. LNCS 1102.

BLS06. M.-P. Béal, S. Lombardy, and J. Sakarovitch. Conjugacy and equivalence of weighted automata and functional transducers. In *Prof. of CSR '06*, pages 58–69. Springer, 2006. LNCS 3967.

BMS13. M. Bonsangue, S. Milius, and A. Silva. Sound and complete axiomatizations of coalgebraic language equivalence. *ACM Transactions on Computational Logic*, 14(1), 2013.

Bor09. M. Boreale. Weighted bisimulation in linear algebraic form. In *Proc. of CONCUR '09*, pages 163–177. Springer, 2009. LNCS 5710.

BPPR14. F. Bonchi, D. Petrisan, D. Pous, and J. Rot. Coinduction up to in a fibrational setting. CoRR abs/1401.6675, 2014.

BR88. J. Berstel and Ch. Reutenauer. *Rational Series and their Languages*. Springer, 1988.

DK13. M. Droste and D. Kuske. Weighted automata, 2013. To appear in *Automata: from Mathematics to Applications*, European Mathematical Society.

DKV09. M. Droste, Werner K., and H. Vogler. *Handbook of Weighted Automata*. Springer, 2009.

Eis03. J. Eisner. Simpler and more general minimization for weighted finite-state automata. In *Proc. of HLT-NAACL '03, Volume 1*, pages 64–71. Association for Computational Linguistics, 2003.

ÉM10. Z. Ésik and A. Maletti. Simulation vs. equivalence. In *Proc. of FCS '10*, pages 119–124, 2010.

FMT05. Gianluigi Ferrari, Ugo Montanari, and Emilio Tuosto. Coalgebraic minimization of hd-automata for the π-calculus using polymorphic types. *Theor. Comput. Sci.*, 331(2-3):325–365, February 2005.

HJS07. I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4:11):1–36, 2007.

HU79. J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages and computation*. Addison Wesley, Reading, Massachusetts, 1979.

KMO$^+$11. S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Language equivalence for probabilistic automata. In *Proc. of CAV '11*, pages 526–540. Springer, 2011. LNCS 6806.

Kro92. D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In *Automata, Languages and Programming*, volume 623 of *LNCS*, pages 101–112. Springer, 1992.

LS89. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing (preliminary report). In *Proc. of POPL '89*, pages 344–352. ACM, 1989.

Moh97. M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:269–311, 1997.

Moh09. M. Mohri. Weighted automata algorithms. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, pages 213–254. Springer, 2009.

RBB$^+$14. J. Rot, F. Bonchi, M. Bonsangue, J. Rutten, D. Pous, and A. Silva. Enhanced coalgebraic bisimulation. *Mathematical Structures in Computer Science*, 2014. to appear.

Rut00. J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

Sta09. S. Staton. Relating coalgebraic notions of bisimulation. In *Proc. of CALCO '09*, pages 191–205. Springer, 2009. LNCS 5728.

UH14. Natsuki Urabe and Ichiro Hasuo. Generic forward and backward simulations III: Quantitative simulations by matrices. In *Proc. of CONCUR '14*. Springer, 2014. LNCS/ARCoSS, to appear.