A Coalgebraic Perspective on Linear Weighted Automata

Filippo Bonchi[a,*], Marcello Bonsangue[b,c], Michele Boreale[d], Jan Rutten[c,e], Alexandra Silva[c]

[a] *CNRS - Laboratoire de l'Informatique du Parallélisme (ENS), 46 Allé d'Italie, 69364 Lyon, France*
[b] *Leiden Institute Advanced Computer Science, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands*
[c] *Centrum Wiskunde & Informatica, Science Park 123, 1098 XG Amsterdam, The Netherlands*
[d] *Dipartimento di Sistemi e Informatica, Universitá di Firenze, Viale Morgagni 65, I-50134 Firenze, Italy*
[e] *Radboud Universiteit Nijmegen, Heyendaalseweg 135, 6525 AJ Nijmegen, The Netherlands*

Abstract

Weighted automata are a generalization of non-deterministic automata where each transition, in addition to an input letter, has also a quantity expressing the weight (e.g. cost or probability) of its execution. As for non-deterministic automata, their behaviours can be expressed in terms of either (weighted) bisimilarity or (weighted) language equivalence.

Coalgebras provide a categorical framework for the uniform study of state-based systems and their behaviours. In this work, we show that coalgebras can suitably model weighted automata in two different ways: coalgebras on Set (the category of sets and functions) characterize weighted bisimilarity, while coalgebras on $Vect$ (the category of vector spaces and linear maps) characterize weighted language equivalence.

Relying on the second characterization, we show three different procedures for computing weighted language equivalence. The first one consists in a generalizion of the usual partition refinement algorithm for ordinary automata. The second one is the backward version of the first one. The third procedure relies on a syntactic representation of rational weighted languages.

1. Introduction

Weighted automata were introduced in Schützenberger's classical paper [33]. They are of great importance in computer science [9], arising in different areas of application, such as speech recognition [23], image compression [2], control theory [16] and quantitative modelling [21, 3]. They can be seen as a generalization of non-deterministic automata, where each transition has a weight associated to it. This weight is an element of a semiring, representing, for example, the cost or probability of taking the transition.

The behaviour of weighted automata is usually given in terms of weighted languages (also called formal power series [32, 5]), that are functions assigning a weight to each finite string $w \in A^*$ over an input alphabet A. For computing the weight given to a certain word, the semiring structure plays a key role: the multiplication of the semiring is used to accumulate the weight of a path by multiplying the weights of each transition in the path, while the addition of the semiring computes the weight of a string w by summing up the weights of the paths labeled with w [20]. Alternatively, the behaviour of weighted automata can be expressed in terms of weighted bisimilarity [7], that is an extension of bisimilarity (for non-deterministic automata) subsuming several kinds of quantitative equivalences such as, for example, probabilistic bisimilarity [17]. As in the case of non-deterministic automata, (weighted) bisimilarity implies strictly (weighted) language equivalence.

Weighted automata still retain non-deterministic behavior, as two different transitions outgoing from the same state may be labelled by the same input action, possibly with different weights. Deterministic weighted automata are of interest because their construction is tightly connected with the existence of minimal automata recognizing the same weighted language. The classical powerset construction for obtaining a language-equivalent deterministic

*Corresponding author

automaton from a non-deterministic one can be generalized to weighted automata, as long as the semiring respects certain restrictions [24, 19]. The states of the determinized automaton are finite "subsets of weighted states" of the original non-deterministic automaton, or, more formally, functions from the set of states to the semiring that are almost everywhere zero. Differently from the classical case, though, the weighted automaton obtained by the powerset construction might be infinite. Usually, one restricts the attention to semirings for which determinization is guaranteed to terminate and produce a finite result, such as locally finite and tropical semirings, and extensions thereof [24, 19].

In this paper, we study *linear weighted automata*, which are deterministic weighted automata where the set of states forms a vector space. A linear weighted automaton can be viewed as the result of determinizing an ordinary weighted automaton with weights in a generic *field*, using a weighted powerset construction. As such, linear weighted automata are typically infinite-state. The key point is that the linear structure of the state-space allows for finite representations of these automata and effective algorithms operating on them.

To be more specific, the goal of the present paper is to undertake a *systematic study of the behavioural equivalences and minimization algorithms for (linear) weighted automata*. To achieve this goal, we will benefit from a coalgebraic perspective on linear weighted automata. The theory of coalgebras offers a unifying mathematical framework for the study of many different types of state-based systems and infinite data structures. Given a functor $\mathcal{G}\colon C \to C$ on a category C, a $\mathcal{G}$-coalgebra is a pair consisting of an object X in C (representing the state space of the system) and a morphism $f{:}X \to \mathcal{G}X$ (determining the dynamics of the system). Under mild conditions, functors $\mathcal{G}$ have a final coalgebra (unique up to isomorphism) into which every $\mathcal{G}$-coalgebra can be mapped via a unique so-called $\mathcal{G}$-homomorphism. The final coalgebra can be viewed as the universe of all possible $\mathcal{G}$-behaviours: the unique homomorphism into the final coalgebra maps every state of a coalgebra to a canonical representative of its behaviour. This provides a general notion of behavioural equivalence ($\approx_{\mathcal{G}}$): two states are equivalent if and only if they are mapped to the same element of the final coalgebra.

Our first contribution in this paper is to recast both weighted bisimilarity and weighted language equivalence in the theory of coalgebras. We see weighted automata for a field $\mathbb{K}$ and alphabet A, as coalgebras of the functor $\mathcal{W} = \mathbb{K} \times \mathbb{K}^{-^A}$ on Set (the category of sets and functions). Concretely, a $\mathcal{W}$-coalgebra consists of a set of states X and a function $\langle o, t\rangle\colon X \to \mathbb{K} \times \mathbb{K}^{X^A}$ where, for each state $x \in X$, $o\colon X \to \mathbb{K}$ assigns an output weight in $\mathbb{K}$ and $t\colon X \to \mathbb{K}^{X^A}$ assigns a function in $\mathbb{K}^{X^A}$. For each symbol $a \in A$ and state $x' \in X$, $t(x)(a)(x')$ is a weight $k \in \mathbb{K}$ representing the weight of a transition from x to x' with label a, in symbols $x \overset{a,k}{\to} x'$. If $t(x)(a)(x') = 0$, then there is no a-labeled transition from x to x'. Note that there could exist several weighted transitions with the same label outgoing from the same state: $x \overset{a,k_1}{\to} x_1,\ x \overset{a,k_2}{\to} x_2,\ \dots,\ x \overset{a,k_n}{\to} x_n$.

Adapting the above reasoning, we model linear weighted automata as coalgebras of the functor $\mathcal{L} = \mathbb{K} \times (-)^A$ on $Vect$ (the category of vector spaces and linear maps). A linear weighted automaton consists of a vector space V and a linear map $\langle o, t\rangle\colon V \to \mathbb{K} \times V^A$ where, as before, $o\colon V \to \mathbb{K}$ defines the output and $t\colon V \to V^A$ the transition structure. More precisely, for each $v \in V$ and $a \in A$, $t(v)(a) = v'$ means that there is a transition from v to v' with label a, in symbols $v \overset{a}{\to} v'$. Note that the transition structure is now "deterministic", since for each v and a there is only one $v' \in V$. When $V = \mathbb{K}^X$, each vector $v \in V$ can be seen as a linear combination of states $x_1, \dots, x_n \in X$, i.e., $v = k_1x_1 + \cdots + k_nx_n$ for some $k_1, \dots, k_n \in \mathbb{K}$. Therefore, the transitions $x \overset{a,k_1}{\to} x_1,\ \dots,\ x \overset{a,k_n}{\to} x_n$ of a weighted automaton correspond to a single transition $x \overset{a}{\to} (k_1x_1 + \cdots + k_nx_n)$ of a linear weighted automaton.

We show that $\approx_{\mathcal{W}}$ (i.e., the behavioural equivalence induced by $\mathcal{W}$) coincides with weighted bisimulation while $\approx_{\mathcal{L}}$ coincides with weighted language equivalence. Determinization is the construction for moving from ordinary weighted automata and weighted bisimilarity to linear weighted automata and weighted language equivalence.

Once we have fixed the mathematical framework, we investigate three different types of algorithms for computing $\approx_{\mathcal{L}}$. These algorithms work under the assumption that the underlying vector space has finite dimension. The first is a forward algorithm that generalizes the usual partition-refinement algorithm for ordinary automata: one starts by decreeing as equivalent states with the same output values, then refines the obtained relation by separating states for which outgoing transitions go to states that are not already equivalent. Linearity of the automata plays a crucial role to ensure termination of the algorithm. Indeed, the equivalences computed at each iteration can be represented as *finite*-dimensional sub-spaces in the given vector space. The resulting descending chain of sub-spaces must therefore converge in a finite number of steps, despite the fact that the state-space itself is infinite. We also show that each iteration of the algorithm coincides with the equivalence generated by each step of the (standard) construction of the final coalgebra via the final sequence. The minimal linear representations of weighted automata over a field was first considered by Schutzenberger [33]. This algorithm was reformulated in a more algebraic and somewhat simplified fashion in Berstel and Reutenauer book [5]. Their algorithm is different from our method, as it is related to the construction of a basis for a subgroup of a free group. Further, no evident connections can be traced between their treatment and the notions of bisimulation and coalgebras.

The second algorithm proceeds in a similar way, but uses a backward procedure. It has been introduced by the third authors together with linear weighted automata [6]. In this case, the algorithm starts from the *complement* – in a precise geometrical sense – of the relation identifying vectors with equal weights. Then it incrementally computes the space of all states that are *backward* reachable from this relation. The largest bisimulation is obtained by taking the complement of this space. The advantage of this algorithm over the previous one is that the size of the intermediate relations is typically much smaller. The presentation of this algorithm in [6] is somewhat more concrete, as there is no attempt at a coalgebraic treatment and the underlying field is fixed to $\mathbb{R}$ (for example, this leads to using orthogonal complements rather than dual spaces and annihilators, which we consider in Section 4). No connection is made with rational series.

Finally, the third algorithm is new and uses the fact that equivalent states are mapped by the unique homomorphism into the same element of the final coalgebra. We characterize the final morphism in terms of so-called rational weighted languages (which are also known as rational formal power series). This characterization is useful for the computation of the kernel of the final homomorphism, which consists of weighted language equivalence. Taking again advantage of the linear structure of our automata, calculating the kernel of the above homomorphism will correspond to solving a linear system of equations.

Structure of the paper. In Section 2 we introduce weighted automata and coalgebras. We also show that $\mathcal{W}$-coalgebras characterize weighted automata and weighted bisimilarity. In Section 3.2, after recalling some preliminary notions of linear algebras, we show that each weighted automaton can be seen as a linear weighted automaton, i.e., an $\mathcal{L}$-coalgebra. This change of perspective allows us to coalgebraically capture weighted language equivalence. In Section 4, we show the forward and the backward algorithm while, in Section 5, we first introduce a syntactic characterization of rational weighted languages and then we shows how to employ it in order to compute $\approx_{\mathcal{L}}$. In Section 6, after summarizing the main results of the paper, we discuss how to extend them to the case of automata with weights in a semiring.

Section 2.2 and Section 4.3 show some interesting minor results that could be safely skipped by the not interested reader. The presentation is self-contained and does not require any prior knowledge on the theory of coalgebras.

2. Weighted Automata as Coalgebras

We introduce weighted automata, weighted bisimilarity and their characterization as coalgebras over Set, the category of sets and functions.

First we fix some notation. We will denote sets by capital letter $X, Y, Z \dots$ and functions by lower case $f, g, h \dots$. Given a set X, id_X is the identity function and, given two functions $f\colon X \to Y$ and $g\colon Y \to Z$, $g \circ f$ is their composition. The product of two sets X, Y is $X \times Y$ with the projection functions $\pi_1{:}X \times Y \to X$ and $\pi_2{:}X \times Y \to Y$. The product of two functions $f_1\colon X_1 \to Y_1$ and $f_2\colon X_2 \to Y_2$ is $f_1 \times f_2$ defined for all $\langle x_1, x_2\rangle \in X_1 \times X_2$ by $(f_1 \times f_2)\langle x_1, x_2\rangle = \langle f(x_1), f(x_2)\rangle$. The disjoint union of X, Y is $X + Y$ with injections $\kappa_1{:}X \to X + Y$ and $\kappa_2{:}Y \to X + Y$. The union of $f_1\colon X_1 \to Y_1$ and $f_2\colon X_2 \to Y_2$ is $f_1 + f_2$ defined for all $z \in X + Y$ by $(f_1 + f_2)(\kappa_i(z)) = k_i((f_i(z)))$ (for $i \in \{1, 2\}$). The set of functions $\varphi\colon Y \to X$ is denoted by X^Y. For $f\colon X_1 \to X_2$, the function $f^Y\colon X_1^Y \to X_2^Y$ is defined for all $\varphi \in X_1^Y$ by $f^Y(\varphi) = \lambda y \in Y. f(\varphi(y))$. The collection of finite subsets of X is denoted by $\mathcal{P}_\omega(X)$ and the emptyset by $\emptyset$. For a set of letters A, A^* denotes the set of all words over A; ϵ the empty word; and $w_1 w_2$ the concatenation of words $w_1, w_2 \in A^*$.

We fix a field $\mathbb{K}$. We use $k_1, k_2, \dots$ to range over elements of $\mathbb{K}$. The sum of $\mathbb{K}$ is denoted by $+$, the product by $\cdot$, the additive identity by 0 and the multiplicative identity by 1. The *support* of a function φ from a set X to a field $\mathbb{K}$ is the set $\{x \in X \mid \varphi(x) \neq 0\}$.

Weighted automata [33, 9] are a generalization of ordinary automata where transitions in addition to an input letter have also a weight in a field $\mathbb{K}$ and each state is not just accepting or rejecting but has an associated output weight in $\mathbb{K}$.

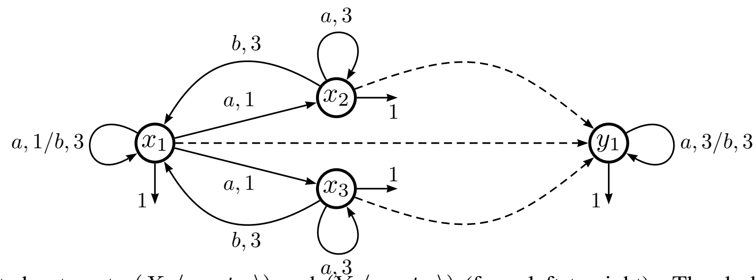


Figure 1: The weighted automata $(X, \langle o_X, t_X \rangle)$ and $(Y, \langle o_Y, t_Y \rangle)$ (from left to right). The dashed arrow denotes the $\mathcal{W}$-homomorphism $h\colon X \to Y$. This induces the equivalence relation $R_h = X \times X$ that equates all the states in X.

Formally, a *weighted automaton* (WA, for short) with input alphabet A is a triple $(X, \langle o, t \rangle)$, where X is a set of states, $o\colon X \to \mathbb{K}$ is an output function associating to each state its output weight and $t\colon X \to (\mathbb{K}^X)^A$ is the transition relation that associates a weight to each transition. We shall use the following notation: $x \overset{a,k}{\to} y$ means that $t(x)(a)(y) = k$. Weight 0 means no transition.

If the set of states is finite, a WA can be conveniently represented in form of matrices. First of all, we have to fix an ordering $(x_1, \ldots, x_n)$ of the set of states X. Then the transition relation t can be represented by a family of matrices $\{T_a\}_{a \in A}$ where each $T_a \in \mathbb{K}^{n \times n}$ is an $\mathbb{K}$-valued square matrix, with $T_a(i, j)$ specifying the value of the a-transition from x_j to x_i, i.e., $t(x_j)(a)(x_i)$. The output weight function o can be represented as an $\mathbb{K}$-valued row vector in $\mathbb{K}^{1 \times n}$ that we will denote by the capital letter O.

For a concrete example, let $\mathbb{K} = \mathbb{R}$ (the field of real numbers) and $A = \{a, b\}$ and consider the weighted automata $(X, \langle o_X, t_X \rangle)$ and $(Y, \langle o_Y, t_Y \rangle)$ in Fig. 1. Their representation as matrix is the following.

$$O_X = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} T_{X_a} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 3 & 0 \\ 1 & 0 & 3 \end{pmatrix} T_{X_b} = \begin{pmatrix} 3 & 3 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad O_Y = \begin{pmatrix} 1 \end{pmatrix} T_{Y_a} = \begin{pmatrix} 3 \end{pmatrix} T_{Y_b} = \begin{pmatrix} 3 \end{pmatrix}$$

Weighted bisimilarity generalizes the abstract semantics of several kind of probabilistic and stochastic systems. This has been introduced by Buchholz in [7] for weighted automata with a finite state space. Here we extend that definition to (possibly infinite-states) automata with *finite branching*, i.e., those $(X, \langle o, t \rangle)$ such that for all $x \in X, a \in A, t(x)(a)(x') \neq 0$ for finitely many x'. This is needed in the following to ensure that $\sum_{x' \in X} t(x)(a)(x')$ is always defined.

Hereafter we will always implicitly refer to weighted automata with finite branching. Moreover, given an $x \in X$ and an equivalence relation $R \subseteq X \times X$ we will write $[x]_R$ to denote the equivalence class of x with respect to R.

Definition 1. Let $(X, \langle o, t \rangle)$ be a weighted automaton. An equivalence relation $R \subseteq X \times X$ is a *weighted bisimulation* if for all $(x_1, x_2) \in R$, it holds that:

1. $o(x_1) = o(x_2)$,
2. $\forall a \in A, x' \in X, \sum_{x'' \in [x']_R} t(x_1)(a)(x'') = \sum_{x'' \in [x']_R} t(x_2)(a)(x'')$.

Weighted bisimilarity (in symbols $\sim_w$) is defined as the largest weighted bisimulation.

For instance, the relation R_h in Fig.1 is a weighted bisimulation.

Now, we will show that weighted automata and weighted bisimilarity can be suitably characterized through *coalgebras* [27].

We first recall some basic definitions about coalgebras. Given a functor $\mathcal{G}\colon C \to C$ on a category C, a $\mathcal{G}$-*coalgebra* is an object X in C together with an arrow $f\colon X \to \mathcal{G}X$. For many categories and functors, such pair (X, f) represents a transition system, the *type* of which is determined by the functor $\mathcal{G}$. Viceversa, many types of transition systems (e.g., deterministic automata, labeled transition systems and probabilistic transition systems) can be captured by a functor.

A $\mathcal{G}$-*homomorphism* from a $\mathcal{G}$-coalgebra (X, f) to a $\mathcal{G}$-coalgebra (Y, g) is an arrow $h : X \to Y$ preserving the transition structure, *i.e.*, such that the following diagram commutes.

$$\begin{array}{ccc} X & \xrightarrow{h} & Y \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ \mathcal{G}X & \xrightarrow[\mathcal{G}h]{} & \mathcal{G}Y \end{array}$$

A $\mathcal{G}$-coalgebra (Ω, ω) is said to be *final* if for any $\mathcal{G}$-coalgebra (X, f) there exists a unique $\mathcal{G}$-homomorphism $[\![-]\!]^{\mathcal{G}}_X : X \to \Omega$. Final coalgebra can be viewed as the universe of all possible $\mathcal{G}$-*behaviours*: the unique homomorphism $[\![-]\!]^{\mathcal{G}}_X : X \to \Omega$ maps every state of a coalgebra X to a canonical representative of its behaviour. This provides a general notion of behavioural equivalence: two states $x_1, x_2 \in X$ are $\mathcal{G}$-*behaviourally equivalent* ($x_1 \approx_{\mathcal{G}} x_2$) iff $[\![x_1]\!]^{\mathcal{G}}_X = [\![x_2]\!]^{\mathcal{G}}_X$ [1].

The functors corresponding to many well known types of systems are shown in [27]. In this section we will show a functor $\mathcal{W}\colon Set \to Set$ such that $\approx_{\mathcal{W}}$ coincides with weighted bisimilarity. In order to do that, we need to introduce the *field valuation functor*.

Definition 2 (Field valuation Functor). Let $\mathbb{K}$ be a field. The field valuation functor $\mathbb{K}^{-}_{\omega} : Set \to Set$ is defined as follows. For each set X, $\mathbb{K}^X_\omega$ is the set of functions from X to $\mathbb{K}$ with finite support. For each function $h\colon X \to Y$, $\mathbb{K}^h_\omega \colon \mathbb{K}^X_\omega \to \mathbb{K}^Y_\omega$ is the function mapping each $\varphi \in \mathbb{K}^X_\omega$ into $\varphi^h \in \mathbb{K}^Y_\omega$ defined, for all $y \in Y$, by

$$\varphi^h(y) = \sum_{x' \in h^{-1}(y)} \varphi(x')$$

Note that the above definition employs only the additive monoid of $\mathbb{K}$, i.e., the element 0 and the $+$ operator. For this reason, such functor is often defined in literature (e.g., in [13]) for commutative monoids instead of fields.

We need two further ingredients. Given a set B, the functor $B \times - \colon Set \to Set$ maps every set X into $B \times X$ and every function $f\colon X \to Y$ into $id_B \times f\colon B \times X \to B \times Y$. Given a finite set A, the functor $-^A\colon Set \to Set$ maps X into X^A and $f\colon X \to Y$ into $f^A\colon X^A \to Y^A$.

Now, the functor corresponding to weighted automata with input alphabet A over the field $\mathbb{K}$ is $\mathcal{W} = \mathbb{K} \times (\mathbb{K}^{-}_{\omega})^A \colon Set \to Set$. Note that every function $f\colon X \to \mathcal{W}(X)$ consists of a pair of functions $\langle o, t\rangle$ with $o\colon X \to \mathbb{K}$ and $t\colon X \to (\mathbb{K}^X_\omega)^A$. Therefore any $\mathcal{W}$-coalgebra (X, f) is a weighted automata $(X, \langle o, t\rangle)$ (and viceversa).

Proposition 1 ([34]) *The functor $\mathcal{W}$ has a final coalgebra.*

In order to show that the equivalence induced by the final $\mathcal{W}$-coalgebra ($\approx_{\mathcal{W}}$) coincides with weighted bisimilarity ($\sim_w$), it is instructive to spell out the notion of $\mathcal{W}$-homomorphism. A function $h\colon X \to Y$ is a $\mathcal{W}$-homomorphism between weighted automata $(X, \langle o_X, t_X\rangle)$ and $(Y, \langle o_Y, t_Y\rangle)$ if the following diagram commutes.

$$\begin{array}{ccc} X & \xrightarrow{\quad h \quad} & Y \\ {\scriptstyle \langle o_X, t_X\rangle}\downarrow & & \downarrow{\scriptstyle \langle o_Y, t_Y\rangle} \\ \mathbb{K} \times (\mathbb{K}^X_\omega)^A & \xrightarrow[id \times (\mathbb{K}^h_\omega)^A]{} & \mathbb{K} \times (\mathbb{K}^Y_\omega)^A \end{array}$$

This means that for all $x \in X, y \in Y, a \in A$,

$$o_X(x) = o_Y(h(x)) \text{ and } \textstyle\sum_{x' \in h^{-1}(y)} t_X(x)(a)(x') = t_Y(h(x))(a)(y).$$

[1] Here we are implicitly assuming that C is a concrete category [1], i.e., there exists a faithfull functor $U\colon C \to Set$. By writing $x_1, x_2 \in X$, we formally mean that $x_1, x_2 \in UX$ and by $[\![x_i]\!]^{\mathcal{G}}_X$, we mean $U([\![-]\!]^{\mathcal{G}}_X)x_i$.

For every $\mathcal{W}$-homomorphism $h\colon (X, \langle o_X, t_X\rangle) \to (Y, \langle o_Y, t_Y\rangle)$, the equivalence relation $R_h = \{(x_1, x_2) \mid h(x_1) = h(x_2)\}$ is a weighted bisimulation. Indeed, by the properties of $\mathcal{W}$-homomorphisms and by definition of R_h, for all $(x_1, x_2) \in R_h$

$$o_X(x_1) = o_Y(h(x_1)) = o_Y(h(x_2)) = o_X(x_2)$$

and for all $a \in A$, for all $y \in Y$

$$\sum_{x'' \in h^{-1}(y)} t_X(x_1)(a)(x'') = t_Y(h(x_1))(a)(y) = t_Y(h(x_2))(a)(y) = \sum_{x'' \in h^{-1}(y)} t_X(x_2)(a)(x'').$$

Trivially, the latter implies that for all $x' \in X$

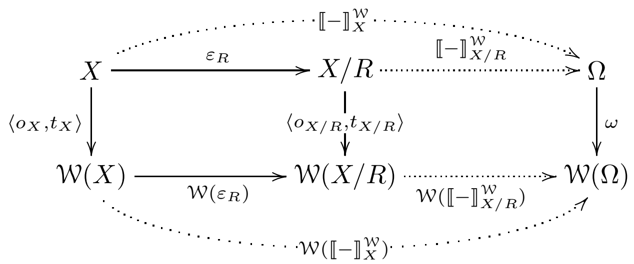
$$\sum_{x'' \in [x']_{R_h}} t_X(x_1)(a)(x'') = \sum_{x'' \in [x']_{R_h}} t_X(x_2)(a)(x'').$$

For an example look at the function h depicted by the dotted arrows in Fig. 1: h is a $\mathcal{W}$-homomorphism and R_h is a weighted bisimulation.

Conversely, every bisimulation R on $(X, \langle o_X, t_X\rangle)$ induces the coalgebra $(X/R, \langle o_{X/R}, t_{X/R}\rangle)$ where X/R is the set of all equivalence classes of X w.r.t. R and $o_{X/R}\colon X/R \to \mathbb{K}$ and $t_{X/R}\colon X/R \to (\mathbb{K}_\omega^{X/R})^A$ are defined for all $x_1, x_2 \in X$, $a \in A$ by

$$o_{X/R}([x_1]_R) = o_X(x_1) \qquad t_{X/R}([x_1]_R)(a)([x_2]_R) = \sum_{x' \in [x_2]_R} t_X(x_1)(a)(x').$$

Note that both $o_{X/R}$ and $t_{X/R}$ are well defined (i.e., independent from the choice of the representative) since R is a weighted bisimulation. Most importantly, the function $\varepsilon_R\colon X \to X/R$ mapping x into $[x]_R$ is a $\mathcal{W}$-homomorphism.



Theorem 1 *Let $(X, \langle o, t\rangle)$ be a weighted automaton and let x_1, x_2 be two states in X. Then, $x_1 \sim_w x_2$ iff $x_1 \approx_{\mathcal{W}} x_2$, i.e., $[\![x_1]\!]_X^{\mathcal{W}} = [\![x_2]\!]_X^{\mathcal{W}}$.*

PROOF. The proof follows almost trivially from the above observations.

If $x_1 \approx_{\mathcal{W}} x_2$, i.e., $[\![x_1]\!]_X^{\mathcal{W}} = [\![x_2]\!]_X^{\mathcal{W}}$, then $(x_1, x_2) \in R_{[\![-]\!]_X^{\mathcal{W}}}$ and $R_{[\![-]\!]_X^{\mathcal{W}}}$ is a weighted bisimulation because $[\![-]\!]_X^{\mathcal{W}}$ is a $\mathcal{W}$-homomorphism. Thus $x_1 \sim_w x_2$

If $x_1 \sim_w x_2$, then there exists a weighted bisimulation R such that $(x_1, x_2) \in R$. Let $(X/R, \langle o_{X/R}, t_{X/R}\rangle)$ and $\varepsilon_R\colon X \to X/R$ be the $\mathcal{W}$-coalgebra and the $\mathcal{W}$-homomorphism described above. Since there exists a unique $\mathcal{W}$-homomorphism from $(X, \langle o_X, t_X\rangle)$ to the final coalgebra, then $[\![-]\!]_X^{\mathcal{W}} = [\![-]\!]_{X/R}^{\mathcal{W}} \circ \varepsilon_R$. Since $\varepsilon_R(x_1) = \varepsilon_R(x_2)$, then $[\![x_1]\!]_X^{\mathcal{W}} = [\![x_2]\!]_X^{\mathcal{W}}$, i.e., $x_1 \approx_{\mathcal{W}} x_2$.

2.1. Weighted language equivalence

The semantics of weighted automata can also be defined in terms of *weighted languages*. A weighted language over A and $\mathbb{K}$ is a function $\sigma\colon A^* \to \mathbb{K}$ assigning to each word in A^* a weight in $\mathbb{K}$. For each WA $(X, \langle o, t\rangle)$, the function $l_X\colon X \to \mathbb{K}^{A^*}$ assigns to each state $x \in X$ its recognized weighted language. For all words $a_1 \dots a_n \in A^*$, it is defined by

$$l_X(x)(a_1 \dots a_n) = \sum\{k_1 \cdot \dots \cdot k_n \cdot k \mid x = x_1 \overset{a_1,k_1}{\to} \dots \overset{a_n,k_n}{\to} x_n \text{ and } o(x_n) = k\}.$$

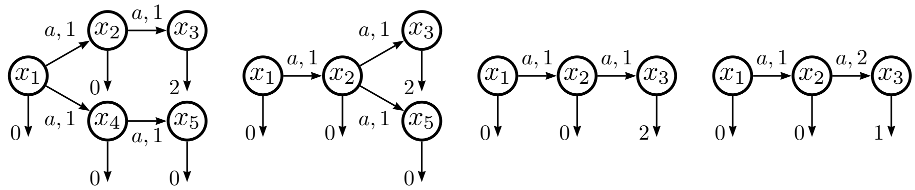


Figure 2: The states x_1 in the above automata recognize the language mapping aa into 2 and the other words into 0. Although they are all language equivalent, they are not bisimilar.

We will often use the following characterization: for all $w \in A^\star$,

$$l_X(x)(w) = \begin{cases} o(x), & \text{if } w = \epsilon; \\ \sum_{x' \in X}(t(x)(a)(x') \cdot l_X(x')(w')), & \text{if } w = aw'. \end{cases}$$

Two states $x_1, x_2 \in X$ are said to be *weighted language equivalent* (denoted by $x_1 \sim_l x_2$) if $l_X(x_1) = l_X(x_2)$. In [7], it is shown that if two states are weighted bisimilar then they are also weighted language equivalent. For completeness, we recall here the proof.

Proposition 2 $\sim_w \subseteq \sim_l$

PROOF. We prove that if R is a weighted bisimulation, then for all $(x_1, x_2) \in R, l_X(x_1) = l_X(x_2)$. We use induction on words $w \in A^*$.

If $w = \epsilon$, then $l_X(x_1)(w) = o(x_1)$ and $l_X(x_2)(w) = o(x_2)$ and $o(x_1) = o(x_2)$ since R is a weighted bisimulation.

If $w = aw'$, then

$$l_X(x_1)(w) = \sum_{x' \in X}(t(x_1)(a)(x') \cdot l_X(x')(w')).$$

By induction hypothesis for all $x'' \in [x']_R$, $l_X(x'')(w') = l_X(x')(w')$. Thus in the above summation we can group all the states $x'' \in [x']_R$ as follows.

$$l_X(x_1)(w) = \sum_{[x']_R \in X/R} \left(l_X(x')(w') \cdot \left(\sum_{x'' \in [x']_R} t(x_1)(a)(x'') \right) \right)$$

Since $(x_1, x_2) \in R$ and R is a weighted bisimulation, the above summation is equivalent to

$$\sum_{[x']_R \in X/R} \left(l_X(x')(w') \cdot \left(\sum_{x'' \in [x']_R} t(x_2)(a)(x'') \right) \right)$$

that, by the previous arguments, is equal to $l_X(x_2)(w)$.

The inverse inclusion does not hold: all the states x_1 in Fig.2 are language equivalent but they are not equivalent according to weighted bisimilarity.

2.2. On the difference between $\mathcal{W}$-bisimilarity and $\mathcal{W}$-behavioural equivalence

We conclude this section with an example showing the difference between $\mathcal{W}$-behavioral equivalence (and hence weighted bisimulation) and another canonical equivalence notion from the theory of coalgebra, namely $\mathcal{W}$-bisimulation. This result is not needed for understanding the next sections, and therefore this sub-section can be safely skipped.

The theory of coalgebras provides an alternative definition of equivalence, namely $\mathcal{G}$-bisimilarity ($\simeq_\mathcal{G}$), that coincides with $\mathcal{G}$-behavioural equivalence whenever the functor $\mathcal{G}$ preserves *weak pullbacks* [27]. In the case of weighted automata, the functor $\mathcal{W}$ does not preserve weak pullbacks and $\simeq_\mathcal{W}$ is strictly included into $\approx_\mathcal{W}$. Since weighted automata are one of the few interesting cases where this phenomenon arises, we now show an example of two states that are in $\approx_\mathcal{W}$, but not in $\simeq_\mathcal{W}$ (the paper [12] was of great inspiration in the construction of this example).

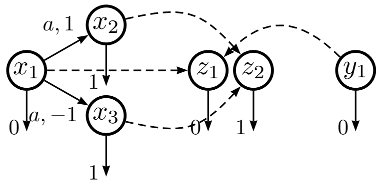
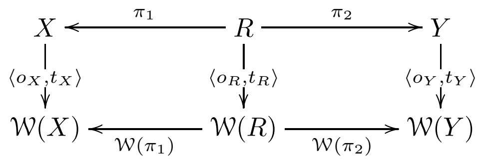


Figure 3: From left to right, three weighted automata over $\mathbb{R}$: $(X, \langle o_X, t_X\rangle)$, $(Z, \langle o_Z, t_Z\rangle)$ and $(Y, \langle o_Y, t_Y\rangle)$. The dashed arrows denotes the $\mathcal{W}$-homomorphisms $h_1 \colon X \to Z$ and $h_2 \colon Y \to Z$. The states x_1 and y_1 are behaviourally equivalent, but they are not $\mathcal{W}$-bisimilar.

First, let us instantiate the general coalgebraic definition of bisimulation and bisimilarity to the functor $\mathcal{W}$. A $\mathcal{W}$-*bisimulation* between two $\mathcal{W}$-coalgebras $(X, \langle o_X, t_X\rangle)$ and $(Y, \langle o_Y, t_Y\rangle)$ is a relation $R \subseteq X \times Y$ such that there exists $\langle o_R, t_R\rangle \colon R \to \mathcal{W}(R)$ making the following diagram commute. The biggest $\mathcal{W}$-bisimulation is called $\mathcal{W}$-bisimilarity ($\simeq_{\mathcal{W}}$).



Note that the actual definition of $\approx_{\mathcal{W}}$ relates the states of a single automaton. We can extend it in order to relate states of possibly distinct automata: given $(X, \langle o_X, t_X\rangle)$ and $(Y, \langle o_Y, t_Y\rangle)$, the states $x \in X$ and $y \in Y$ are equivalent w.r.t. $\approx_{\mathcal{W}}$ iff $[\![x]\!]_X^{\mathcal{W}} = [\![y]\!]_Y^{\mathcal{W}}$.

Consider now the coalgebras in Fig.3: $x_1 \approx_{\mathcal{W}} y_1$, but $x_1 \not\simeq_{\mathcal{W}} y_1$. For the former, it is enough to observe that the function h_1 and h_2 (represented by the dashed arrows) are $\mathcal{W}$-homomorphisms, and by uniqueness of $[\![-]\!]^{\mathcal{W}}$: $[\![x_1]\!]_X^{\mathcal{W}} = [\![h_1(x_1)]\!]_Z^{\mathcal{W}} = [\![z_1]\!]_Z^{\mathcal{W}} = [\![h_2(y_1)]\!]_Z^{\mathcal{W}} = [\![y_1]\!]_Y^{\mathcal{W}}$. For $x_1 \not\simeq_{\mathcal{W}} y_1$, note that there exists no $R \subseteq X \times Y$ that is a $\mathcal{W}$-bisimulation and such that $(x_1, y_1) \in R$. Since x_2 and x_3 are both different from y_1 (their output values are different), then neither (x_2, y_1) nor (x_3, y_1) can belong to a bisimulation. Thus the only remaining relation on $X \times Y$ is the one equating just x_1 and y_1, i.e., $R = \{(x_1, y_1)\}$. But this is not a $\mathcal{W}$-bisimulation since there exists no $\langle o_R, t_R\rangle$ making the leftmost square of the above diagram commute. In order to understand this fact, note that $\pi_1^{-1}(x_2) = \emptyset$ and $\pi_1^{-1}(x_3) = \emptyset$. Thus for all possible choices of $\langle o_R, t_R\rangle$, the function $\mathcal{W}(\pi_1) \circ \langle o_R, t_R\rangle$ maps (x_1, y_1) into a pair $\langle k, \varphi\rangle$ where $\varphi(a)(x_2) = 0$ and $\varphi(a)(x_3) = 0$. On the other side of the square, we have that $\langle o_X, t_X\rangle \circ \pi_1(x_1, y_1) = \langle o_X(x_1), t_X(x_1)\rangle$ and $t_X(x_1)(a)(x_2) = 1$ and $t_X(x_1)(a)(x_3) = -1$.

3. Linear Weighted Automata as Linear Coalgebras

In this section we will introduce linear weighted automata as coalgebras for an endofunctor $\mathcal{L} \colon Vect \to Vect$, where $Vect$ is the category of vector spaces and linear maps over a field $\mathbb{K}$. The goal of this change is to characterize weighted language equivalence as the behavioural equivalence induced by the final $\mathcal{L}$-coalgebra.

3.1. Preliminaries

First we fix some notations and recall some basic facts on vector spaces and linear maps. We use $v_1, v_2, \ldots$ to range over vectors and $V, W \ldots$ to range over vector spaces on a field $\mathbb{K}$. Given a vector space V, a vector $v \in V$ and a $k \in \mathbb{K}$, the scalar product is denoted by $k \cdot v$ (or kv for short). The *space spanned* by an I-indexed family of vectors $B = \{v_i\}_{i \in I}$ is the space $\mathrm{span}(B)$ of all v such that

$$v = k_1 v_{i_1} + k_2 v_{i_2} + \cdots + k_n v_{i_n}$$

where for all j, $v_{i_j} \in B$. In this case, we say that v is a *linear combination* of the vectors in B. A set of vectors is *linearly independent* if none of its elements can be expressed as the linear combination of the remaining ones. A *basis* for the space V is a linearly independent set of vectors that spans the whole V. All the basis of V have the same cardinality which is called the *dimension* of V (denoted by $dim(V)$). If $(v_1, \ldots v_n)$ is a basis for V, then each vector

$v \in V$ is equal to $k_1 v_1 + \cdots + k_n v_n$ for some $k_1, \ldots, k_n \in \mathbb{K}$. For this reason, each vector v can be represented as a $n \times 1$-column vector

$$v = \begin{pmatrix} k_1 \\ \vdots \\ k_n \end{pmatrix}$$

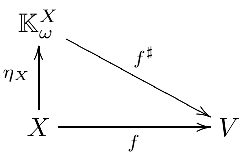
We use $f, g, \ldots$ to range over linear maps. Identity and composition of maps are denoted as usual. If $B_V = (v_1, \ldots v_n)$ and $B_W = (w_1 \ldots w_m)$ are, respectively, the basis for the vector spaces V and W, then every linear map $f\colon V \to W$ can be represented as $m \times n$-matrix. Indeed, for each $v \in V$, $v = k_1 v_1 + \cdots + k_n v_n$ and $f(v) = k_1 f(v_1) + \cdots + k_n f(v_n)$, by linearity of f. For each v_i, $f(v_i)$ can be represented as $m \times 1$ column vector by taking as basis B_W. Thus the matrix corresponding to f (w.r.t. B_V and B_W) is the one having as i-th column the vector corresponding to $f(v_i)$. In this paper we will use capital letters $F, G \ldots$ to denote the matrices corresponding to linear maps $f, g \ldots$ in lower case. By multiplying the matrix F for the vector v (in symbols, $F \times v$) we can compute $f(v)$. More generally, matrix multiplication corresponds to composition of linear maps, in symbols:

$$g \circ f = G \times F$$

The product of two vector spaces V, W is written as $V \times W$, and the product of two linear maps f_1, f_2 is $f_1 \times f_2$, defined as for functions. It will be clear from the context whether $\times$ refer to multipliaction of matrix or product of spaces (or maps). Given a set X, and a vector space V, the set V^X (i.e., the set of functions $\varphi\colon X \to V$) carries a vector space structure where sum and scalar product are defined point-wise. Hereafter we will use V^X to denote both the vector space and the underlying carrier set. Given a linear map $f\colon V_1 \to V_2$, the linear map $f^X\colon V_1^X \to V_2^X$ is defined as for functions. If A is a finite set we can conveniently think V^A as the product of V with itself for $|A|$-times ($|A|$ is the cardinality of A). A linear map $f\colon U \to V^A$ can be decomposed in a family of maps indexed by A, $f = \{f_a\colon U \to V\}_{a \in A}$, such that for all $u \in U$, $f_a(u) = f(u)(a)$.

For a set X, the set $\mathbb{K}^X_\omega$ (i.e., the set of all finite support functions $\varphi\colon X \to \mathbb{K}$) carries a vector space where sum and scalar product are defined in the obvious way. This is called the *free vector space* generated by X and can be thought of as the space spanned by the elements of X: each vector $k_1 x_{i_1} + k_2 x_{i_2} + \cdots + k_n x_{i_n}$ corresponds to a function $\varphi\colon X \to \mathbb{K}$ such that $\varphi(x_{i_j}) = k_j$ and for all $x \notin \{x_{i_j}\}$, $\varphi(x) = 0$; conversely, each finite support function φ corresponds to a vector $\varphi(x_{i_1})x_i + \varphi(x_{i_2})x_{i_2} + \cdots + \varphi(x_{i_n})x_{i_n}$.

A fundamental property holds in the free vector space generated by X: for all functions f from X to the carrier-set of a vector space V, there exists a linear map $f^\sharp\colon \mathbb{K}^X_\omega \to V$ that is called the *linearization* of f. For all $v \in \mathbb{K}^X_\omega$, $v = k_1 x_{i_1} + k_2 x_{i_2} + \cdots + k_n x_{i_n}$ and $f^\sharp(v) = k_1 f(x_{i_1}) + k_2 f(x_{i_2}) + \cdots + k_n f(x_{i_n})$.



Note that $f^\sharp$ is the only linear map such that $f = f^\sharp \circ \eta_X$, where $\eta_X(x)$ is the function assigning 1 to x and 0 to all the other elements of X.

The *kernel* $\ker(f)$ of a linear map $f\colon V \to W$ is the subspace of V containing all the vectors $v \in V$ such that $f(v) = 0$. The *image* $\mathrm{im}(f)$ of f is the subspace of W containing all the $w \in W$ such that $w = f(v)$ for some $v \in V$. If V has finite dimension, the kernel and the image of f are related by the following equation:

$$\dim(V) = \dim(\ker(f)) + \dim(\mathrm{im}(f))\,. \tag{1}$$

Given two vector spaces V_1 and V_2, their intersection $V_1 \cap V_2$ is still a vector space, while their union $V_1 \cup V_2$ is not. Instead of union we consider the coproduct of vector spaces: we write $V_1 + V_2$ to denote the space $\mathrm{span}(V_1 \cup V_2)$ (note that in the category of vector spaces, product and coproduct coincide).

3.2. From Weighted Automata to Linear Weighted Automata

We have now all the ingredients to introduce linear weighted automata and a coalgebraic characterization of weighted language equivalence.

Definition 3 (LWA). A *linear weighted automaton* (LWA, for short) with input alphabet A over the field $\mathbb{K}$ is a coalgebra for the functor $\mathcal{L} = \mathbb{K} \times -^A \colon Vect \to Vect$.

More concretely [6], a LWA is a triple $(V, \langle o, t \rangle)$, where V is a vector space (representing the states space), $o \colon V \to \mathbb{K}$ is a linear map associating to each state its output weight and $t \colon V \to V^A$ is a linear map that for each input $a \in A$ associates a next state (i.e., a vector) in V. We will write $v_1 \xrightarrow{a} v_2$ for $t(v_1)(a) = v_2$.

The behaviour of linear weighted automata is expressed in terms of weighted languages. The *language recognized* by a vector $v \in V$ of a LWA $(V, \langle o, t \rangle)$ is defined for all words $a_1 \dots a_n \in A^*$ as $[\![v]\!]_V^{\mathcal{L}}(a_1 \dots a_n) = o(v_n)$ where v_n is the vector reached from v through $a_1 \dots a_n$, i.e., $v \xrightarrow{a_1} \dots \xrightarrow{a_n} v_n$. We will often use the following (more compact) definition: for all $w \in A^\star$,

$$[\![v]\!]_V^{\mathcal{L}}(w) = \begin{cases} o(v), & \text{if } w = \epsilon; \\ [\![t(v)(a)]\!]_V^{\mathcal{L}}(w'), & \text{if } w = aw'. \end{cases}$$

Here we use the notation $[\![-]\!]_V^{\mathcal{L}}$ because this is the unique $\mathcal{L}$-homomorphism into the final $\mathcal{L}$-coalgebra. In Section 3.3, we will provide a proof for this fact and we will also discuss the exact correspondence with the function l_X introduced in Section 2.

Given a weighted automaton $(X, \langle o, t \rangle)$, we can build a linear weighted automaton $(\mathbb{K}_\omega^X, \langle o^\sharp, t^\sharp \rangle)$, where $\mathbb{K}_\omega^X$ is the free vector space generated by X and $o^\sharp$ and $t^\sharp$ are the linearizations of o and t. If X is finite, we can represent $t^\sharp$ and $o^\sharp$ by the same matrices that we have introduced in the previous section for t and o. By fixing an ordering $x_1, \dots, x_n$ of the states in X, we have a basis for $\mathbb{K}_\omega^X$, i.e., every vector $v \in \mathbb{K}_\omega^X$ is equal to $k_1 x_1 + \dots + k_n x_n$ and it can be represented as an $n \times 1$-column vector. The values $t^\sharp(v)(a)$ and $o^\sharp(v)$ can be computed via matrix multiplication as $T_a \times v$ and $O \times v$.

For a concrete example, look at the weighted automaton $(X, \langle o_X, t_X \rangle)$ in Fig. 1. The corresponding linear weighted automaton $(\mathbb{R}_\omega^X, \langle o_X^\sharp, t_X^\sharp \rangle)$ has as state space the space of all the linear combinations of the states in X (i.e., $\{k_1 x_1 + k_2 x_2 + k_3 x_3 \mid k_i \in \mathbb{R}\}$). The function $o_X^\sharp$ maps $v = k_1 x_1 + k_2 x_2 + k_3 x_3$ into $k_1 o_X(x_1) + k_2 o_X(x_2) + k_3 o_X(x_3)$, i.e., $k_1 + k_2 + k_3$. By exploiting the correspondence between functions and vectors in $\mathbb{K}_\omega^X$ (discussed in Section 3.1), we can write $t_X^\sharp(v)(a) = k_1 t_X(x_1)(a) + k_2 t_X(x_2)(a) + k_3 t_X(x_3)(a)$ that is $k_1(x_1 + x_2 + x_3) + k_2 3x_2 + k_3 3x_3$ and $t_X^\sharp(v)(b) = k_1 3x_1 + k_2 3x_1 + k_3 3x_1$. This can be conveniently expressed in terms of matrix multiplication.

$$o_X^\sharp(v) = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix} \; t_X^\sharp(v)(a) = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 3 & 0 \\ 1 & 0 & 3 \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix} \; t_X^\sharp(v)(b) = \begin{pmatrix} 3 & 3 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix}$$

A linear map $h \colon V \to W$ is an $\mathcal{L}$-homomorphism between LWA $(V, \langle o_V, t_V \rangle)$ and $(W, \langle o_W, t_W \rangle)$ if the following diagram commutes.

$$\begin{array}{ccc} V & \xrightarrow{\quad h \quad} & W \\ {\scriptstyle \langle o_V, t_V \rangle} \downarrow & & \downarrow {\scriptstyle \langle o_W, t_W \rangle} \\ \mathbb{K} \times V^A & \xrightarrow[id \times h^A]{} & \mathbb{K} \times W^A \end{array}$$

This means that for all $v \in V, a \in A$, $o_V(v) = o_W(h(v))$ and $h(t_V(v)(a)) = t_W(h(v))(a)$. If V and W have finite dimension, then we can represent all the morphisms of the above diagram as matrices. In this case, the above diagram commutes if and only if for all $a \in A$,

$$O_V = O_W \times H \qquad H \times T_{V_a} = T_{W_a} \times H$$

where T_{V_a} and T_{W_a} are the matrix representation of t_V and t_W for any $a \in A$.

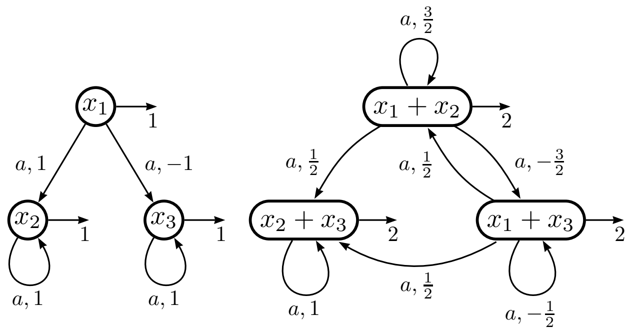


Figure 4: The weighted automata $(X, \langle o_X, t_X \rangle)$ (left) and $(Y, \langle o_Y, t_Y \rangle)$ (right). The corresponding linear weighted automata $(\mathbb{R}^X_\omega, \langle o^\sharp_X, t^\sharp_X \rangle)$ and $(\mathbb{R}^Y_\omega, \langle o^\sharp_Y, t^\sharp_Y \rangle)$ are isomorphic.

For a function $h\colon X \to Y$, the function $\mathbb{K}^h\colon \mathbb{K}^X \to \mathbb{K}^Y$ (formally introduced in Definition 2) is a linear map. Note that if h is a $\mathcal{W}$-homomorphism between the WA $(X, \langle o_X, t_X \rangle)$ and $(Y, \langle o_Y, t_Y \rangle)$, then $\mathbb{K}^h$ is an $\mathcal{L}$-homomorphism between the LWA $(\mathbb{K}^X, \langle o^\sharp_X, t^\sharp_X \rangle)$ and $(\mathbb{K}^Y, \langle o^\sharp_Y, t^\sharp_Y \rangle)$. For an example, look at the $\mathcal{W}$-homomorphism $h\colon (X, \langle o_X, t_X \rangle) \to (Y, \langle o_Y, t_Y \rangle)$ represented by the dotted arrows in Fig. 1. The linear map $\mathbb{R}^h\colon \mathbb{R}^X \to \mathbb{R}^Y$ is represented by the matrix $H = (1\ 1\ 1)$ and it is an $\mathcal{L}$-homomorphism between $(\mathbb{R}^X, \langle o^\sharp_X, t^\sharp_X \rangle)$ and $(\mathbb{R}^Y, \langle o^\sharp_Y, t^\sharp_Y \rangle)$. This can be easily checked by showing that $O_X = O_Y \times H$, $H \times T_{X_a} = T_{Y_a} \times H$ and $H \times T_{X_b} = T_{Y_b} \times H$.

Note that two different weighted automata can *represent* the same (up to isomorphism) linear weighted automaton. As an example, look at the weighted automata $(X, \langle o_X, t_X \rangle)$ and $(Y, \langle o_Y, t_Y \rangle)$ in Fig. 4. They represent, respectively, the linear weighted automata $(\mathbb{R}^X_\omega, \langle o^\sharp_X, t^\sharp_X \rangle)$ and $(\mathbb{R}^Y_\omega, \langle o^\sharp_Y, t^\sharp_Y \rangle)$ that are isomorphic. The transitions and the output functions for the two automata are described by the following matrices.

$$T_{X_a} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} O_X = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \qquad T_{Y_a} = \begin{pmatrix} \frac{3}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ -\frac{3}{2} & 0 & -\frac{1}{2} \end{pmatrix} O_Y = \begin{pmatrix} 2 & 2 & 2 \end{pmatrix}$$

Note that T_{X_a} and T_{Y_a} are *similar*, i.e., they represent the same linear map. This can be immediately checked by showing that $T_{Y_a} = j^{-1} \circ t_{X_a} \circ j$, where $j\colon \mathbb{R}^Y \to \mathbb{R}^X$ is the isomorphic map representing the change of bases form $Y = (x_1 + x_2, x_2 + x_3, x_3 + x_1)$ to $X = (x_1, x_2, x_3)$ and $j^{-1}\colon \mathbb{R}^X \to \mathbb{R}^Y$ is its inverse. The matrix representation of j and j^{-1} is the following.

$$J = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \qquad J^{-1} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Also O_X and O_Y represents the same map in different bases. Indeed, $O_Y = O_X \times J$.

At this point, it is easy to see that the linear isomorphism $j^{-1}\colon \mathbb{R}^X \to \mathbb{R}^Y$ is an $\mathcal{L}$-homomorphism, because $O_X = O_X \times J \times J^{-1} = O_Y \times J^{-1}$ and $J^{-1} \times T_{X_a} = J^{-1} \times T_{X_a} \times J \times J^{-1} = T_{Y_a} \times J^{-1}$. Analogously for $j\colon \mathbb{R}^Y \to \mathbb{R}^X$.

3.3. Language equivalence and final $\mathcal{L}$-coalgebra

We introduce the final $\mathcal{L}$-coalgebra and we show that the behavioural equivalence $\approx_{\mathcal{L}}$, induced by the functor $\mathcal{L}$, coincides with weighted language equivalence.

The set of all weighted languages $\mathbb{K}^{A^*}$ carries a vector space structure: the sum of two languages $\sigma_1, \sigma_2 \in \mathbb{K}^{A^*}$ is the language $\sigma_1 + \sigma_2$ defined for each word $w \in A^*$ as $\sigma_1 + \sigma_2(w) = \sigma_1(w) + \sigma_2(w)$; the product of a language σ for a scalar $k \in \mathbb{K}$ is $k\sigma$ defined as $k\sigma(w) = k \cdot \sigma(w)$; the element 0 of $\mathbb{K}^{A^*}$ is the language mapping each word into the 0 of $\mathbb{K}$.

The *empty function* $\epsilon\colon \mathbb{K}^{A^*} \to \mathbb{K}$ and the *derivative function* $d\colon \mathbb{K}^{A^*} \to (\mathbb{K}^{A^*})^A$ are defined for all $\sigma \in \mathbb{K}^{A^*}$, $a \in A$ as

$$\epsilon(\sigma) = \sigma(\epsilon) \quad d(\sigma)(a) = \sigma_a$$

where $\sigma_a{:}A^* \to \mathbb{K}$ denotes the *a-derivative* of σ that is defined for all $w \in A^*$ as

$$\sigma_a(w) = \sigma(aw).$$

Proposition 3 *The maps* $\epsilon\colon \mathbb{K}^{A^*} \to \mathbb{K}$ *and* $d\colon \mathbb{K}^{A^*} \to (\mathbb{K}^{A^*})^A$ *are linear.*

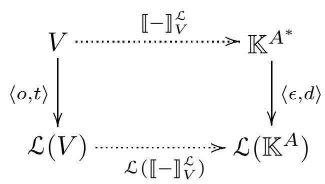
PROOF. We show the proof for d. The one for ϵ is analogous.

Let σ_1, σ_2 be two weighted languages in $\mathbb{K}^{A^*}$. Now for all $a \in A, w \in A^*$, $d(\sigma_1 + \sigma_2)(a)(w) = \sigma_1 + \sigma_2(aw) = \sigma_1(aw) + \sigma_2(aw) = d(\sigma_1)(a)(w) + d(\sigma_2)(a)(w)$.

Let k be a scalar in $\mathbb{K}$ and σ be a weighted language in $\mathbb{K}^{A^*}$. Now for all $a \in A, w \in A^*$, $k \cdot d(\sigma)(a)(w) = k \cdot \sigma(aw) = d(k\sigma)(a)(w)$.

Since $\mathbb{K}^{A^*}$ is a vector space and since ϵ and d are linear maps, $(\mathbb{K}^{A^*}, \langle \epsilon, d\rangle)$ is an $\mathcal{L}$-coalgebra. The following theorem shows that it is final.

Theorem 2 (finality) *From every $\mathcal{L}$-coalgebra $(V, \langle o, t\rangle)$ there exists a unique $\mathcal{L}$-homomorphism into $(\mathbb{K}^{A^*}, \langle \epsilon, d\rangle)$.*



PROOF. The only function making the above diagram commutes is $[\![-]\!]^{\mathcal{L}}_V$, i.e., the function mapping each vector $v \in V$ into the weighted language that it *recognizes*. Hereafter we show that $[\![-]\!]^{\mathcal{L}}_V$ is a linear map.

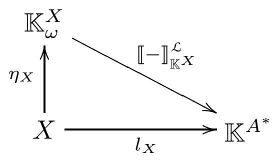
By induction on w, we prove that for all $v_1, v_2 \in V$, for all $w \in A^*$, $[\![v_1 + v_2]\!]^{\mathcal{L}}_V(w) = [\![v_1]\!]^{\mathcal{L}}_V(w) + [\![v_2]\!]^{\mathcal{L}}_V(w)$.

Suppose that $w = \epsilon$. Then $[\![v_1 + v_2]\!]^{\mathcal{L}}_V(\epsilon) = o(v_1 + v_2)$. Since o is a linear map, this is equal to $o(v_1) + o(v_2) = [\![v_1]\!]^{\mathcal{L}}_V(\epsilon) + [\![v_2]\!]^{\mathcal{L}}_V(\epsilon)$.

Now suppose that $w = aw'$. Then $[\![v_1 + v_2]\!]^{\mathcal{L}}_V(aw') = [\![t(v_1 + v_2)(a)]\!]^{\mathcal{L}}_V(w')$. Since t is a linear map, this is equal to $[\![t(v_1)(a) + t(v_2)(a)]\!]^{\mathcal{L}}_V(w')$ that (by induction hypothesis) is equal to $[\![t(v_1)(a)]\!]^{\mathcal{L}}_V(w') + [\![t(v_2)(a)]\!]^{\mathcal{L}}_V(w') = [\![v_1]\!]^{\mathcal{L}}_V(aw') + [\![v_2]\!]^{\mathcal{L}}_V(aw')$.

We can proceed analogously for the scalar product.

Thus, two vectors $v_1, v_2 \in V$ are $\mathcal{L}$-behaviourally equivalent ($v_1 \approx_{\mathcal{L}} v_2$) iff they recognize the same weighted language (as defined in Section 3.2). Proposition 4 below shows that $[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega} : \mathbb{K}^X_\omega \to \mathbb{K}^{A^*}$ is the linearization of the function $l_X\colon X \to \mathbb{K}^{A^*}$ (defined in Section 2) or, in other words, is the only linear map making the following diagram commute.



Lemma 1 *Let $(X, \langle o, t\rangle)$ be a* WA *and $(\mathbb{K}^X_\omega, \langle o^\sharp, t^\sharp\rangle)$ be the corresponding linear weighted automaton. Then for all $x \in X$, $l_X(x) = [\![x]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}$.*

PROOF. We prove it by induction on $w \in A^*$.

If $w = \epsilon$, then $l_X(x)(w) = o_X(x) = o^\sharp_X(x) = [\![x]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w)$.

If $w = aw'$, then $[\![x]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w) = [\![t^\sharp(x)(a)]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w')$. Note that by definition, $t^\sharp(x)(a) = \sum_{x' \in X} t(x)(a)(x')x'$, thus the latter is equal to

$$\Big[\!\!\Big[\sum_{x' \in X} t(x)(a)(x') \cdot x' \Big]\!\!\Big]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w')$$

which, by linearity of $[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}$, coincides with

$$\sum_{x' \in X} t(x)(a)(x') \cdot [\![x']\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w').$$

By induction hypothesis $[\![x']\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w') = l_X(x')(w')$ and thus the above coincides with

$$\sum_{x' \in X} t(x)(a)(x') \cdot l_X(x')(w')$$

that is $l_X(x)(w)$.

Proposition 4 *Let $(X, \langle o, t\rangle)$ be a* WA *and $(\mathbb{K}^X_\omega, \langle o^\sharp, t^\sharp\rangle)$ be the corresponding linear weighted automaton. Then, for all $v = k_1 x_{i_1} + \cdots + k_n x_{i_n}$, $[\![v]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega} = k_1 l_X(x_{i_1}) + \cdots + k_n l_X(x_{i_n})$.*

PROOF. By induction on $w \in A^*$.

If $w = \epsilon$, then $[\![v]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w) = o^\sharp(v)$. Since $o^\sharp$ is a linear map and X is a base for $\mathbb{K}^X_\omega$, $o^\sharp(v) = k_1 o(x_{i_1}) + \cdots + k_n o(x_{i_n})$. For all j, $l_X(x_{i_j})(\epsilon) = o(x_{i_j})$, thus $k_1 o(x_{i_1}) + \cdots + k_n o(x_{i_n}) = k_1 l_X(x_{i_1})(w) + \cdots + k_n l_X(x_{i_n})(w)$.

If $w = aw'$, then $[\![v]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w) = [\![t^\sharp(v)(a)]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w')$. Since $t^\sharp$ is linear and X is a base for $\mathbb{K}^X_\omega$, then $t^\sharp(v)(a) = k_1 t(x_{i_1})(a) + \cdots + k_n t(x_{i_n})(a)$. For all j,

$$t(x_{i_j})(a) = \sum_{x' \in X} \left(t(x_{i_j})(a)(x') \cdot x' \right),$$

thus $[\![t^\sharp(v)(a)]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w')$ is equal to

$$\left[\!\!\left[k_1 \sum_{x' \in X} \left(t(x_{i_j})(a)(x') \cdot x' \right) + \cdots + k_n \sum_{x' \in X} \left(t(x_{i_j})(a)(x') \cdot x' \right) \right]\!\!\right]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w')$$

which, by linearity of $[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}$, is equal to

$$k_1 \sum_{x' \in X} \left(t(x_{i_j})(a)(x') \cdot [\![x']\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w') \right) + \cdots + k_n \sum_{x' \in X} \left(t(x_{i_j})(a)(x') \cdot [\![x']\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w') \right).$$

By induction hypothesis $[\![x']\!]^{\mathcal{L}}_{\mathbb{K}^X_\omega}(w') = l_X(x')(w')$ and thus the latter coincides with

$$k_1 \sum_{x' \in X} \left(t(x_{i_j})(a)(x') \cdot l_X(x')(w') \right) + \cdots + k_n \sum_{x' \in X} \left(t_X(x_{i_j})(a)(x') \cdot l_X(x')(w') \right).$$

By definition, $l_X(x_{i_j})(w) = \sum_{x' \in X} \left(t(x_{i_j})(a)(x') \cdot l_X(x')(w') \right)$ and thus we can concisely express the above formula as

$$k_1 l_X(x_1)(w) + \cdots + k_n l_X(x_n)(w).$$

3.4. Linear Bisimulations and Subspaces

We now introduce a convenient characterization of language equivalence by means of *linear weighted bisimulations*. Differently form ordinary (weighted) bisimulations, these can be seen both as relations and as subspaces. The latter characterization will be exploited in the next section for defining an algorithm for checking language equivalence.

First, we show how to represent relations over a vector space V as sub-spaces of V, following [35, 6].

Definition 4 (linear relations). Let U be a sub-space of V. The binary relation R_U over V is defined by

$$v_1 \; R_U \; v_2 \text{ if and only if } v_1 - v_2 \in U.$$

A relation R is *linear* if there is a subspace U such that $R = R_U$.

Note that a linear relation is a total equivalence relation on V. Let now R be *any* binary relation over V. There is a canonical way of turning R into a linear relation, which we describe in the following. The *kernel* of R (in symbols $\ker(R)$) is the set $\{v_1 - v_2 \mid v_1 \, R \, v_2\}$. The *linear extension* of R, denoted R^ℓ, is defined by: $v_1 \, R^\ell \, v_2$ if and only if $(v_1 - v_2) \in \mathrm{span}(\ker(R))$.

Lemma 2 *Let U be a sub-space of V, then* $\ker(R_U) = U$.

According to the above lemma, a linear relation R is completely described by its kernel, which is a sub-space, that is

$$v_1 \, R \, v_2 \quad \text{if and only if} \quad (v_1 - v_2) \in \ker(R) \,. \tag{2}$$

Conversely, for any sub-space $U \subseteq V$ there is a corresponding linear relation R_U whose kernel is U. Hence, without loss of generality, *we can identify linear relations on V with sub-spaces of V*. For example, by slight abuse of notation, we can write $v_1 \, U \, v_2$ instead of $v_1 \, R_U \, v_2$; and conversely, we will sometimes denote by R the sub-space $\ker(R)$. The context will be sufficient to tell whether we are actually referring to a linear relation or to the corresponding sub-space (kernel). Note that the sub-space $\{0\}$ corresponds to the identity relation on V, that is $R_{\{0\}} = Id_V$. In fact: $v_1 \, Id_V \, v_2$ iff $v_1 = v_2$ iff $v_1 - v_2 = 0$. Similarly, the space V itself corresponds to $R_V = V \times V$.

We are now ready to define linear weighted bisimulation. This definition relies on the familiar step-by-step game played on transitions, plus an initial condition requiring that two related states have the same output weight. We call this form of bisimulation *linear* to stress the difference with the one introduced in Definition 1.

Definition 5 (linear weighted bisimulation). Let $(V, \langle o, t \rangle)$ be a linear weighted automaton. A linear relation $R \subseteq V \times V$ is a *linear weighted bisimulation* if for all $(v_1, v_2) \in R$, it holds that:

(1) $o(v_1) = o(v_2)$,
(2) $\forall a \in A, t(v_1)(a) \; R \; t(v_2)(a)$.

For a concrete example, consider the automaton $(\mathbb{R}^X_\omega, \langle o^\sharp_X, t^\sharp_X \rangle)$ in Fig 4. The relation $R = \{(x_2, x_3)\}$ is not linear, because $U = \{x_2 - x_3\}$ is not a subspace of $\mathbb{R}^X_\omega$. However, we can turn R into a linear relation by employing its kernel $\ker(R) = \{x_2 - x_3\}$. The linear extension of R is $R^\ell = \{(k_1 x_1 + k_2 x_2 + k_3 x_3, k'_1 x_1 + k'_2 x_2 + k'_3 x_3) \mid k_1 = k'_1 \text{ and } k_2 + k_3 = k'_2 + k'_3\}$. It is easy to see that R^ℓ is a linear weighted bisimulation.

The following lemma provides a characterization of linear weighted bisimulation as a subspace. Let us say that a sub-space U is *f-invariant* if $f(U) \subseteq U$. Bisimulations are transition-invariant relations that refine the kernel of the output map o.

Lemma 3 *Let $(V, \langle o, t \rangle)$ be a* LWA *and R be linear relation over V. R is a linear weighted bisimulation if and only if*

(1) $R \subseteq \ker(o)$,
(2) R is t_a-invariant for each $a \in A$.

This lemma will be fundamental in the next section for defining an algorithm computing the greatest linear weighted bisimulation. In the remainder of this section, we show that the greatest linear weighted bisimulation coincides with the kernel of the final map $[\![-]\!]^{\mathcal{L}}_V$. More generally, the kernel of each $\mathcal{L}$-homomorphism is a linear weighted bisimulation R and, viceversa, for each linear weighted bisimulation R there exists an $\mathcal{L}$-homomorphism whose kernel is R.

Proposition 5 *Let $(V, \langle o_V, t_V \rangle)$ be a* LWA. *If $f \colon V \to W$ is an $\mathcal{L}$-homomorphism (for some* LWA *$(W, \langle o_W, t_W \rangle)$) then $\ker(f)$ is a linear weighted bisimulation. Conversely, if R is a linear weighted bisimulation for $(V, \langle o, t \rangle)$, then there exists a* LWA *$(W, \langle o_W, t_W \rangle)$ and an $\mathcal{L}$-homomorphism $f \colon V \to W$ such that $\ker(f) = R$.*

PROOF. First, we suppose that $f \colon V \to W$ is an $\mathcal{L}$-homomorphism and we prove that $\ker(f)$ satisfies (1) and (2) of Lemma 3. Take a vector $v \in \ker(f)$. Thus, $f(v) = 0$ and, since o_W and t_W are linear maps, $o_W(f(v)) = 0$ and $t_W(f(v))(a) = 0$ for all $a \in A$. Since f is an $\mathcal{L}$-homomorphism, we have that (1) $o_V(v) = o_W(f(v)) = 0$, i.e., $\ker(f) \subseteq \ker(o_V)$ and (2) $f(t_V(v)(a)) = t_W(f(v))(a) = 0$ meaning that $t_V(v)(a) \in \ker(f)$, i.e., $\ker(f)$ is t_{V_a}-invariant.

In order to prove the second part, we need to recall *quotient spaces* and *quotient maps* from [14]. Given a subspace U of V, the equivalence class of v w.r.t. U is $[v]_U = \{v + u \mid u \in U\}$. Note that $v_1 \in [v_2]_U$ if and only if $v_1 R_U v_2$. The quotient space V/U is the space of all equivalence classes $[v]_U$ where scalar product $k[v]_U$ is defined as $[kv]_U$ and the sum $[v_1]_U + [v_2]_U$ as $[v_1 + v_2]_U$. It is easy to check that these operations are well-defined (i.e., independent from the choice of the representative) and turn V/U into a vector space where the element 0 is U. Most importantly, the quotient function $\varepsilon_U : V \to V/U$ mapping each vector v into $[v]_U$ is a linear map such that $\ker(\varepsilon_U) = U$.

Now, let U be the subspace corresponding to the linear weighted bisimulation R. We can take $W = V/U$ and we define o_W as $o_W([v]_U) = o_V(v)$ and t_W as $t_W([v]_U)(a) = [t(v)(a)]_U$. Note that both o_W and t_W are well defined: for all $v' \in [v]_U = \{v+u \mid u \in U\}$, $o_W(v') = o_W(v)$ (since $o_V(u) = 0$ for all $u \in U$) and $t_W(v')(a) \in [t_W(v)(a)]_U$ (since $t_V(u)(a) \in U$ for all $u \in U$). It is also easy to check that they are linear.

Finally, we take $f : V \to W$ as ε_U and with the previous definition of o_W and t_W is trivial to check that ε_U is an $\mathcal{L}$-homomorphism. As said above, its kernel is U.

Theorem 3 *Let $(V, \langle o, t\rangle)$ be a* LWA *and let $[\![-]\!]^{\mathcal{L}}_V : V \to \mathbb{K}^{A^*}$ be the unique $\mathcal{L}$-morphism into the final coalgebra. Then $\ker([\![-]\!]^{\mathcal{L}}_V)$ is the largest linear weighted bisimulation on V.*

PROOF. First of all, note that by the first part of Proposition 5, $\ker([\![-]\!]^{\mathcal{L}}_V)$ is a linear weighted bisimulation.

Then suppose that R is a linear weighted bisimulation. By the second part of Proposition 5, there exists a LWA $(W, \langle o_W, t_W\rangle)$ and an $\mathcal{L}$-homomorphism $f : V \to W$ such that $R = \ker(f)$. Now note that, since $(W, \langle o_W, t_W\rangle)$ is an $\mathcal{L}$-coalgebra there exists an $\mathcal{L}$-homomorphism $[\![-]\!]^{\mathcal{L}}_W : W \to \mathbb{K}^{A^*}$ to the final coalgebra. Since the composition of two $\mathcal{L}$-homomorphisms is still an $\mathcal{L}$-homomorphism, also $[\![-]\!]^{\mathcal{L}}_W \circ f : V \to \mathbb{K}^{A^*}$ is an $\mathcal{L}$-homomorphism. Since $[\![-]\!]^{\mathcal{L}}_V$ is the unique $\mathcal{L}$-homomorphism from V to $\mathbb{K}^{A^*}$, then $[\![-]\!]^{\mathcal{L}}_W \circ f = [\![-]\!]^{\mathcal{L}}_V$. Finally, $R = \ker(f) \subseteq \ker([\![-]\!]^{\mathcal{L}}_W \circ f) = \ker([\![-]\!]^{\mathcal{L}}_V)$.

The characterization of bisimulations as subspaces seems to be possible in $Vect$ and not in Set because the former category is *abelian* [10]: every map has a kernel that is a subspace and every subspace is the kernel of some map. We leave as future work to study (at a more general level) the categorical machinery allowing to characterize bisimulations as subspaces.

4. Linear Partition Refinement

In the previous section, we have shown that weighted language equivalence ($\sim_l$) can be seen as the largest linear weighted bisimulation. In this section, we exploit this characterization in order to provide a "partition refinement" algorithm that allows to compute $\sim_l$. We will examine below two versions of the algorithm, a forward version (Section 4.1) and a backward one (Section 4.2). The former is straightforward but computationally not very convenient; the latter is more convenient, although it requires the introduction of some extra machinery. In both cases, we must restrict to LWA's where the state space is finite dimension.

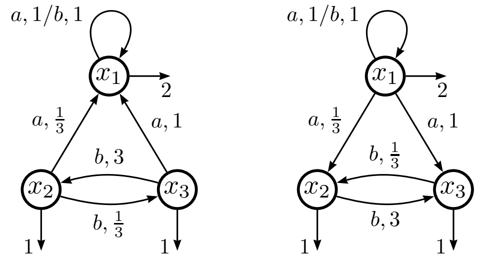
4.1. A forward algorithm

Lemma 3 suggests that, in order to compute the largest linear weighed bisimulation for a LWA $(V, \langle o, t\rangle)$, one might start from $\ker(o)$ and refine it until the condition (2) given in the lemma is satisfied. This is indeed the case.

Proposition 6 (partition refinement, forward version) *Let $(V, \langle o, t\rangle)$ be a* LWA. *Consider the sequence $(R_i)_{i\geq 0}$ of sub-spaces of V defined inductively by*

$$R_0 \;=\; \ker(o) \qquad R_{i+1} \;=\; R_i \cap \bigcap_{a\in A} t(R_i)(a)^{-1}$$

where $t(R_i)(a)^{-1}$ is the space $\{v \in V \mid t(v)(a) \in R_i\}$. Then there is $j \leq \dim(V)$ such that $R_{j+1} = R_j$. The largest linear weighted bisimulation is $\approx_{\mathcal{L}} = R_j$.



$$O = \begin{pmatrix} 2 & 1 & 1 \end{pmatrix} \; T_a = \begin{pmatrix} 1 & \frac{1}{3} & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \; T_b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & \frac{1}{3} & 0 \end{pmatrix} \; {}^{\mathrm{t}}T_a = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{3} & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \; {}^{\mathrm{t}}T_b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & \frac{1}{3} \\ 0 & 3 & 0 \end{pmatrix}$$

Figure 5: A weighted automata $(V, \langle o, t\rangle)$ (left) and its reversed $(V, \langle o, {}^{\mathrm{t}}t\rangle)$ (right).

PROOF. The R_i's form a descending chain of sub-spaces of V. The corresponding dimensions form a non-increasing sequence, hence the existence of j as required is obvious. That R_j is a bisimulation follows by applying Lemma 3: indeed, it is obvious that (1) $\ker(o) \supseteq R_j$, while as to (2) we have that, since $R_{j+1} = R_j$, then $R_j \cap \bigcap_{a \in A} t(R_j)(a)^{-1} = R_j$, i.e., for all $a \in A$, $t(R_j)(a) \subseteq R_j$.

We finally show that any linear weighted bisimulation R is included in R_j. We do so by proving that for each i, $R \subseteq R_i$, thus, in particular $R \subseteq R_j$. We proceed by induction on i. Again by Lemma 3, we know that $R_0 = \ker(o) \supseteq R$. Assume now $R \subseteq R_i$. For each action a, by Lemma 3 we have that $t(R)(a) \subseteq R$, which implies $R \subseteq \{v \in R_i \mid \forall a \in A,\ t(v)(a) \in R_i\} = R_{i+1}$.

Concretely, the algorithm iteratively computes a basis B_i for each space R_i. This can be done by solving systems of linear equations expressing the constraints in the definition of R_i. Since the backward algorithm presented in the next section is computationally more efficient, we avoid to give further details about its implementation and we show, as an example, the algorithm at work with the linear automata $(V, \langle o, t\rangle)$ in Fig.5.

At the beginning, we compute a basis for $R_0 = \ker(o)$. This is

$$B_0 = \left\{ \begin{pmatrix} -\frac{1}{2} \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -\frac{1}{2} \\ 0 \\ 1 \end{pmatrix} \right\}.$$

In the first iteration, we compute one basis for the space $t(R_0)(a)^{-1}$ and one for the space $t(R_0)(b)^{-1}$. These are respectively

$$B_1^a = \left\{ \begin{pmatrix} -\frac{1}{3} \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \right\} \text{ and } B_1^b = \left\{ \begin{pmatrix} -\frac{1}{6} \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -\frac{3}{2} \\ 0 \\ 1 \end{pmatrix} \right\}.$$

Then, R_1 is given by the intersection $R_0 \cap t(R_0)(a)^{-1} \cap t(R_0)(b)^{-1}$. A basis for R_1 is

$$B_1 = \left\{ \begin{pmatrix} -2 \\ 3 \\ 1 \end{pmatrix} \right\}.$$

In the second iteration, we compute one basis for the space $t(R_1)(a)^{-1}$ and one for the space $t(R_1)(b)^{-1}$. These are respectively

$$B_2^a = \left\{ \begin{pmatrix} -\frac{1}{3} \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \right\} \text{ and } B_2^b = \left\{ \begin{pmatrix} -2 \\ 3 \\ 1 \end{pmatrix} \right\}.$$

Then, R_2 is the intersection $R_1 \cap t(R_1)(a)^{-1} \cap t(R_0)(b)^{-1}$. A basis for R_2 is

$$B_2 = \left\{ \begin{pmatrix} -2 \\ 3 \\ 1 \end{pmatrix} \right\}$$

that is equal to B_1. Since $R_1 = R_2$ the algorithm terminates and returns R_1. Now, in order to check if two vectors $v_1, v_2 \in V$ accept the same weighted language (i.e., $v_1 \approx_{\mathcal{L}} v_2$), we have to look if $v_1 - v_2$ belongs to R_1. For instance, $x_1 \approx_{\mathcal{L}} \frac{3}{2}x_2 + \frac{1}{2}x_3$ because $x_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3 \in R_1$.

We note that $\ker(o)$ is in general a large sub-space: since $o\colon V \to \mathbb{K}$ with $\dim(\mathbb{K}) = 1$, by virtue of equation (1) we have that $\dim(\ker(o)) \geq \dim(V) - 1$. This might be problematic in the actual computation of the basis of $\approx_{\mathcal{L}}$. We present an alternative version in the next subsection which will avoid this problem.

4.2. A backward algorithm

Two well-known concepts from linear algebra will be relied upon to describe the basic operations of the backward algorithm. More precisely, annihilators will be used to describe the complement of a relation, while transpose maps will be used to describe the operation of "reversing arrows" in an automaton. These operations are carried out within the *dual space* of V. So we start by reviewing the concept of dual space; an in-depth treatment can be found in e.g. [14].

Let $\mathbb{K}$ be any field and V a vector space over $\mathbb{K}$. The *dual space* of V, denoted $V^\star$, is the set of all linear maps $V \to \mathbb{K}$, with $\mathbb{K}$ seen as a 1-dimensional vector space. The elements of $V^\star$ are often called *functionals* and we use $\psi_1, \psi_2, \ldots$ to range over them. The sum of two functionals $\psi_1 + \psi_2$ and the scalar multiplication $k \cdot \psi$ (for $k \in \mathbb{K}$) are defined point-wise as expected, and turn $V^\star$ into a vector space over $\mathbb{K}$. We will denote functional application $\psi(v)$ as $[v, \psi]$, the bracket notation intending to emphasize certain analogies with inner products. Fix an ordered basis $B = (v_1, ..., v_n)$ of V and consider $B^\star = (v_1^\star, ..., v_n^\star)$, where the functionals $v_i^\star$ are specified by $[v_j, v_i^\star] = \delta_{ij}$ for each i and j. Here, δ_{ij} denotes the Kronecker symbol, which equals 1 if $i = j$ and 0 otherwise. It is easy to check that $B^\star$ forms a basis of $V^\star$, referred to as the *dual basis* of B. Hence $\dim(V^\star) = \dim(V)$. In particular, the morphism $(-)^\star\colon V \to V^\star$ that sends each v_i into $v_i^\star$ is an isomorphism between V and $V^\star$. A crucial definition is that of transpose morphism.

Definition 6 (transpose linear map). Let $f\colon V \to V$ be a linear map. We let the *transpose of* f be the endomorphism ${}^{\mathrm{t}}f\colon V^\star \to V^\star$ defined for all $\psi \in V^\star$ as ${}^{\mathrm{t}}f(\psi) = \psi \circ f$.

It is easy to check that that if F is the matrix representing f in V w.r.t. to B, then the transpose matrix ${}^{\mathrm{t}}F$ represents ${}^{\mathrm{t}}f$ in $V^\star$ w.r.t. $B^\star$, whence the terminology and the notation. It is quite expected that, by taking the transpose twice one gets back the original morphism. This is in fact the case, although one has to take care of identifying things up to isomorphism. Denote by $V^{\star\star}$ the space $(V^\star)^\star$, called double dual of V. There is a natural isomorphism i between V and $V^{\star\star}$, given by $i\colon v \mapsto [v, -]$ (note that this isomorphism does not depend on the choice of a basis). In the sequel, we shall freely identify V and $V^{\star\star}$ up to this isomorphism, i.e. identify v and $[v, -]$ for each $v \in V$. With this identification, one has that ${}^{\mathrm{t}}({}^{\mathrm{t}}f) = f$.

We need another concept from duality theory. Given a subspace U of V, we denote by U^o the *annihilator* of U, the subset of functionals that vanish on U.

Definition 7 (annihilator). For any $U \subseteq V$, we let $U^o = \{\psi \in V^\star \mid [u, \psi] = 0 \text{ for each } u \in U\}$.

Once again, the notation makes the analogy with inner products explicit. We use the following properties of annihilators, where U, W are a sub-spaces of V: (i) U^o is a sub-space of $V^\star$; (ii) $(-)^o$ reverses inclusions, i.e. if $U \subseteq W$ then $W^o \subseteq U^o$; (iii) $(-)^o$ is an involution, that is $(U^o)^o = U$ up to the natural isomorphism between V and its double dual. These three properties suggest that U^o can be regarded as a *complement*, or negation, of U seen as a relation. Another useful property is: (iv) $\dim(U^o) + \dim(U) = \dim(V)$. Transpose morphisms and annihilators are connected via the following property, which is crucial to the development of the algorithm. It basically asserts that f-invariance of R corresponds to ${}^{\mathrm{t}}f$-invariance of the complementary relation represented by R^o.

Lemma 4 *Let U be a sub-space of V and f be an endomorphism on V. If U is f-invariant then U^o is ${}^{\mathrm{t}}f$-invariant.*

We are now ready to give the backward version of the partition refinement algorithm. An informal preview of the algorithm is as follows. Rather than computing directly the sub-space representing $\approx_{\mathcal{L}}$, the algorithm computes the sub-space representing the complementary relation. To this end, the algorithm starts from a relation R_0 that is the complement of the relation identifying vectors with equal weights, then incrementally computes the space of all states that are *backward* reachable from R_0. The largest bisimulation is obtained by taking the complement of this space. Geometrically, "going backward" means working with the transpose transition functions ${}^{\mathrm{t}}t_a$ rather than with t_a. Taking the complement of a relation actually means taking its annihilator. This essentially leads one to work within $V^\star$ rather than V. Recall that $U + W$ denotes $\mathrm{span}(U \cup W)$.

Theorem 4 (partition refinement, backward version) *Let $(V, \langle o, t\rangle)$ be a* LWA*. Consider the sequence $(R_i)_{i\geq 0}$ of sub-spaces of $V^\star$ inductively defined by:*

$$R_0 \quad = \quad \ker(o)^o \qquad R_{i+1} \quad = \quad R_i + \textstyle\sum_{a\in A} {}^{\mathrm{t}}t_a(R_i)\,. \tag{3}$$

Then there is $j \leq \dim(L)$ such that $R_{j+1} = R_j$. The largest $\mathcal{L}$-bisimulation $\approx_{\mathcal{L}}$ is R_j^o, modulo the natural isomorphism between V and $V^{\star\star}$.

PROOF. Since $R_0 \subseteq R_1 \subseteq R_2 \subseteq \cdots \subseteq V^\star$, the sequence of the dimensions of these spaces is non-decreasing. As a consequence, for some $j \leq \dim(V^\star) = \dim(L)$, we get $\dim(R_j) = \dim(R_{j+1})$. Since $R_j \subseteq R_{j+1}$, this implies $R_j = R_{j+1}$.

We next show that R_j^o is an $\mathcal{L}$-bisimulation. Indeed, by the properties of annihilators and up to the natural isomorphism: (1) $\ker(o)^o \subseteq R_j$ implies $(\ker(o)^o)^o = \ker(o) \supseteq R_j^o$. Moreover: (2) for any $a \in A$, ${}^{\mathrm{t}}t_a(R_j) \subseteq {}^{\mathrm{t}}t_a(R_j) + R_j \subseteq R_{j+1} = R_j$ implies, by Lemma 4, that ${}^{\mathrm{t}}({}^{\mathrm{t}}t_a(R_j^o)) = t_a(R_j^o) \subseteq R_j^o$; by (a), (b) and Lemma 3, we conclude that R_j^o is an $\mathcal{L}$-bisimulation.

We finally show that any $\mathcal{L}$-bisimulation R is included in R_j^o. We do so by proving that for each i, $S \subseteq R_i^o$, thus, in particular $S \subseteq R_j^o$. We proceed by induction on i. Again by Lemma 3, we know that $R_0^o = \ker(o) \supseteq R$. Assume now $R \subseteq R_i^o$, that is, $R^o \supseteq R_i$. For each action a, by Lemma 3 we have that $t_a(R) \subseteq R$, which implies ${}^{\mathrm{t}}t_a(R^o) \subseteq R^o$ by Lemma 4. Hence $R^o \supseteq {}^{\mathrm{t}}t_a(R^o) \supseteq {}^{\mathrm{t}}t_a(R_i)$, where the last inclusion stems from $R^o \supseteq R_i$. Since this holds for each a, we have that $R^o \supseteq \sum_a {}^{\mathrm{t}}t_a(R_i) + R_i = R_{i+1}$. Taking the annihilator of both sides reverses the inclusion and yields the wanted result.

We note that what is being "refined" in the algorithm above are not, of course, the sub-spaces R_i, but their complements: $R_0^o \supseteq R_1^o \supseteq \cdots \supseteq R_j^o = \approx_{\mathcal{L}}$. In particular, we start with a "small" space R_0^o of dimension ≤ 1: this may represent an advantage in a practical implementation of the algorithm.

To conclude the section, we briefly discuss some practical aspects involved in the implementation of the algorithm. By virtue of (2), checking $u \approx_{\mathcal{L}} v$, for any pair of vectors v_1 and v_2, is equivalent to checking $v_1 - v_2 \in \ker(\approx_{\mathcal{L}})$. This can be done by first computing a basis of $\approx_{\mathcal{L}}$ and then checking for linear (in)dependence of $v_1 - v_2$ from this basis. Alternatively, and more efficiently, one can check whether $v_1 - v_2$ is in R_j^o, or, more explicitly, whether $[v_1 - v_2, \psi] = 0$ for each $\psi \in R_j$. This reduces to showing whether $[v_1 - v_2, \psi] = 0$ for each $\psi \in B_j$, where B_j is a basis for R_j. Thus, our task reduces to computing such a basis. To do so, fix any basis B of V and let O and T_a ($a \in A$) be the row-vector and matrices representing the weight and transition functions of the LWA in this basis. The concrete computations are carried out representing vectors and functionals in this basis.

1. Compute a basis B_0 of R_0. As already discussed, $\dim(\ker(o)) \geq \dim(V) - 1$, hence $\dim(\ker(o)^o) \leq 1$. It is readily checked that $o \in \ker(o)^o$, thus $\ker(o)^o$ is spanned by o. We thus set $B_0 = \{o\}$. With respect to the chosen basis B, B_0 is represented by $\{O\}$.
2. For each $i \geq 0$, compute a basis B_{i+1} of R_{i+1}. This can be obtained by incrementally joining to B_i the functionals ${}^{\mathrm{t}}t_a(\psi)$, for $a \in A$ and $\psi \in B_i$, that are linearly independent from previously joined functionals. With respect to the basis B, ${}^{\mathrm{t}}t_a(\psi)$ is represented by $\Psi \times T_a$, where Ψ is the row-vector representing ψ; checking linear independence of ${}^{\mathrm{t}}t_a(\psi)$ means hence checking linear independence of $\Psi \times T_a$ from previously joined row-vectors.

After $j \leq n$ iterations, one finds a set B_j such that $B_{j+1} = B_j$: this is the basis of R_j. We illustrate this algorithm in the example below.

Consider the LWA $(V, \langle o, t\rangle)$ on the left of Figure 5. At the beginning we can set $B_0 = \{O\}$. Next, we apply the algorithm to build the B_i's. Manually, the computation of the vectors $\Psi T_a = {}^{\mathrm{t}}({}^{\mathrm{t}}T_a {}^{\mathrm{t}}\Psi)$ can be carried out by looking at the transitions of the WA with arrows reversed (in the right of Figure 5). Doing so, we first get $OT_a = (2\ \frac{2}{3}\ 2)$ and $OT_b = (2\ \frac{1}{3}\ 3)$. Note that OT_b is not linearly independent from the other vectors: $OT_b = -(2\ 1\ 1) + 2(2\ \frac{2}{3}\ 2)$. Thus $B_1 = \{(2\ 1\ 1),\ (2\ \frac{2}{3}\ 2)\}$. In the second iteration, we compute $(2\ \frac{2}{3}\ 2)T_a = (2\ \frac{2}{3}\ 2)$ and $(2\ \frac{2}{3}\ 2)T_b = (2\ \frac{2}{3}\ 2)$ and thus $B_2 = \{(2\ 1\ 1),\ (2\ \frac{2}{3}\ 2)\}$ that is equal to B_1.

The functionals represented by vectors in B_1 are a basis of $(\approx_{\mathcal{L}})^o$. As an example, let us check that $x_1 \approx_{\mathcal{L}} \frac{3}{2}x_2 + \frac{1}{2}x_3$. To this purpose, note that the difference vector $x_1 - \frac{3}{2}x_2 - \frac{1}{2}x_3$ annihilates B_1, that is

$$[\begin{pmatrix} 1 \\ -\frac{3}{2} \\ -\frac{1}{2} \end{pmatrix}, u] = 0$$

for each $u \in B_1$, which is equivalent to $2x_1 \approx_{\mathcal{L}} \frac{3}{2}x_2 + \frac{1}{2}x_3$.

It is quite easy to give an upper bound on the cost of the backward algorithm, in terms of the number of sum and product operations in the underlying field. Let n be the dimension of V. Each time we join a new vector $v = \Psi \times T_a$ to the basis B, we have a cost of $O(n^2)$ for vector-matrix multiplication, plus a cost of $O(n^3)$ for checking linear independence of v from B, for a predominant cost of $O(n^3)$. Since the operation of joining a vector to the basis cannot be done more than n times, we have a global cost of $O(n^4)$. In the case $|A| = 1$, one can adapt the Arnoldi's iteration algorithm [31] to compute B, which takes $O(n^3)$ operations. It is not clear whether this algorithm can be adapted also to the case $|A|\rangle 1$. In practical cases, the transition matrices tend to be sparse, and the number of iterations after which the algorithms stops may be much less than n. By adopting suitable representations for sparse matrices, these circumstances can be used to lower considerably the practical complexity of the algorithm.

4.3. The final sequence and the forward algorithm

The theory of coalgebras also provides a way of constructing final coalgebras by means of *final sequences* (often referred in literature as terminal sequences) [4]. Many important algorithms for computing behavioural equivalences (such as [18]) can be abstractly described in terms of final sequences.

In this section, we describe the relationship between the forward algorithm (in Proposition 6) and the final sequence of the functor $\mathcal{L}$. The latter is the cochain

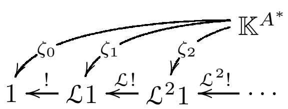
$$1 \xleftarrow{!} \mathcal{L}1 \xleftarrow{\mathcal{L}!} \mathcal{L}^2 1 \xleftarrow{\mathcal{L}^2 !} \ldots$$

where $\mathcal{L}^{n+1}1$ is $\mathcal{L} \circ (\mathcal{L}^n 1)$, $\mathcal{L}^0 1 = 1$ is the final vector space $\{0\}$, and ! is the unique morphism from $\mathcal{L}1$ to 1.

Let A^*_n be the set of all words $w \in A^*$ with *length* smaller than n. For each n, $\mathcal{L}^n 1$ is isomorphic to $\mathbb{K}^{A^*_n}$, i.e., the space of functions from A^*_n to $\mathbb{K}$. Indeed, for $n = 1$, $\mathcal{L}1$ is by definition $\mathbb{K} \times 1^A = \mathbb{K}$ that is isomorphic to the space of functions from $A^*_1 = \{\epsilon\}$ to $\mathbb{K}$; and for $n + 1$, each $\langle k, \sigma\rangle \in \mathbb{K} \times \mathcal{L}^n(1)^A = \mathcal{L}^{n+1}1$ can be seen as function $A^*_{n+1} \to \mathbb{K}$ mapping ϵ into k and aw (for $a \in A$ and $w \in A^*_n$) into $\sigma(a)(w)$.

For $\sigma \colon A^*_m \to \mathbb{K}$ and $n \leq m$, the *n-restriction* of σ is $\sigma \upharpoonright n \colon A^*_n \to \mathbb{K}$ defined as σ, but in a restricted domain. The morphism $\mathcal{L}^n! \colon \mathcal{L}^{n+1}1 \to \mathcal{L}^n 1$ maps each σ into $\sigma \upharpoonright n$.

The limit of this cochain is $\mathbb{K}^{A^*}$ together with the maps $\zeta_n \colon \mathbb{K}^{A^*} \to \mathcal{L}^n 1$ that assign to each weighted language σ its n-restriction $\sigma \upharpoonright n$.



Every $\mathcal{L}$-coalgebra $(V, \langle o, t\rangle)$ determines a cone $!^n \colon V \to \mathcal{L}^n 1$ as follows:

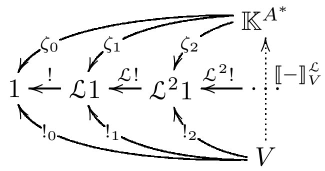
- $!^0 \colon V \to 1$ is the unique morphism to the final vector space 1,

- $!^{n+1}\colon V \to \mathcal{L}^{n+1}1 = \mathcal{L}(!^n) \circ \langle o, t\rangle$.

The latter can be more concretely defined for all $v \in V$ and $w \in \mathbb{K}^{A^*_{n+1}}$ as

$$!^{n+1}(v)(w) = \begin{cases} o(v), & \text{if } w = \epsilon; \\ !^n(t(v)(a))(w'), & \text{if } w = aw'. \end{cases}$$

Note that the final morphism $[\![-]\!]^{\mathcal{L}}_V\colon V \to \mathbb{K}^{A^*}$ (mapping each $v \in V$ in the language that it recognizes) is the unique function such that for all n, $\zeta_n \circ [\![-]\!]^{\mathcal{L}}_V = !^n$.



Recall that the $\mathcal{L}$-behavioural equivalence on $(V, \langle o, t\rangle)$ is the kernel of $[\![-]\!]^{\mathcal{L}}_V$. The forward algorithm computes it, by iteratively computing the kernel of the morphisms $!^n$.

Proposition 7 *Let $(V, \langle o, t\rangle)$ be a* LWA*. Let R_n be the relation computed by the forward algorithm (Proposition 6). Let $!^n\colon V \to \mathcal{L}^n 1$ be the morphisms described above. Then for all natural numbers n, $R_n = \ker(!^{n+1})$.*

PROOF. First of all, note that the kernel of $!^0\colon V \to 1$ is the whole V. The kernel of $!^{n+1}$ is the space of $v \in V$ such that $!^{n+1}(v)(w) = 0$ for all the words $w \in A^*_{n+1}$, i.e.,

$$\ker(!^{n+1}) = \{v \in V \mid o(v) = 0 \text{ and } \forall a \in A,\ t(v)(a) \in \ker(!^n)\}.$$

By induction on n, we prove that $\ker(!^{n+1}) = R_n$.

For $n = 0$, note that $\ker(!^1) = \{v \in V \mid o(v) = 0 \text{ and } \forall a \in A,\ t(v)(a) \in \ker(!_0)\}$. Since $\ker(!^0) = V$, $\ker(!^1) = \{v \in V \mid o(v) = 0\} = R_0$.

As induction hypothesis suppose that $\ker(!^n) = R_{n-1}$. Then $\ker(!^{n+1}) = \{v \in V \mid o(v) = 0 \text{ and } \forall a \in A,\ t(v)(a) \in R_{n-1}\} = R_n$.

This result can be seen as an alternative proof of the soundness of the forward algorithm. Indeed, if R_j is the result of the algorithm, for all $k \geq j$, $R_k = R_j$, i.e., $\ker(!^k) = \ker(!^j)$. Thus $R_j = \bigcap_n \ker(!^n)$ and, by definition of $!^n$, $\bigcap_n \ker(!^n) = \ker([\![-]\!]^{\mathcal{L}}_V)$.

5. Weighted languages and rationality

We recall from Section 3 that a linear weighted automaton (LWA) is a coalgebra for the functor $\mathcal{L} = \mathbb{K} \times -^A$, i.e., it consists of a vector space V and a linear map $\langle o, t\rangle\colon V \to \mathbb{K} \times V^A$. We saw in Theorem 2 that the final homomorphism

$$[\![-]\!]^{\mathcal{L}}_V\colon V \to \mathbb{K}^{A^*}$$

maps every vector $v \in V$ to the weighted language $[\![v]\!]^{\mathcal{L}}_V$ that is accepted by v. Moreover, the kernel of this morphism is weighted language equivalence ($\approx_{\mathcal{L}}$) that, when V is finite dimension, can be computed via the linear partition refinement algorithm (shown in Section 4).

The languages in $\mathbb{K}^{A^*}$ that are accepted by LWA with finite dimension states spaces are called *rational* weighted languages (which are also known as rational formal power series) and they can be syntactically represented by a language of expressions [28].

In this section, we shall directly characterise $[\![-]\!]^{\mathcal{L}}_V$ by showing the expression of $[\![v]\!]^{\mathcal{L}}_V$ for each $v \in V$ (Theorem 5). Then we shall employ this characterization for computing $\approx_{\mathcal{L}}$.

We will first treat the special case of LWA's over a one letter alphabet $|A| = 1$. Next we will show how to treat the general case of an arbitrary (finite) alphabet.

We note that for the case of $|A| = 1$, the functor $\mathcal{L}$ is isomorphic to

$$\mathcal{L}(V) = \mathbb{K} \times V^A \cong \mathbb{K} \times V$$

Moreover, the final $\mathcal{L}$-coalgebra is isomorphic to the set of streams over the field $\mathbb{K}$:

$$\mathbb{K}^{A^*} \cong \mathbb{K}^\omega$$

Therefore we shall proceed by recalling from [30] the basics of stream calculus and linear stream differential equations, in Subsections 5.1 and 5.2. Next we shall characterise the final homomorphism, for the case $|A| = 1$, in Subsection 5.3. Building on [28], we shall finally generalise these results for finite alphabets, in Subsection 5.4.

5.1. Recalling the basics of stream calculus

We define the set of *streams* over the field $\mathbb{K}$ by

$$\mathbb{K}^\omega = \{\sigma \mid \sigma : \mathbb{N} \to \mathbb{K}\}$$

(where $\mathbb{N}$ is the set of natural numbers).

We often denote elements $\sigma \in \mathbb{K}^\omega$ by $\sigma = (\sigma(0), \sigma(1), \sigma(2), \ldots)$. We define the *stream derivative* of a stream σ by $\sigma' = (\sigma(1), \sigma(2), \sigma(3), \ldots)$, and the *initial value* of σ by $\sigma(0)$.

For $k \in \mathbb{K}$, we define the constant stream

$$[k] = (k, 0, 0, 0, \ldots)$$

which we often denote again by k. Another constant stream is

$$\mathcal{X} = (0, 1, 0, 0, 0, \ldots)$$

For $\sigma, \tau \in \mathbb{K}^\omega$ and $n \in \omega$, the operations of *sum* and (convolution) *product* are given by

$$(\sigma + \tau)(n) = \sigma(n) + \tau(n) \ , \qquad (\sigma \times \tau)(n) = \sum_{i=0}^{n} \sigma(i) \cdot \tau(n-i)$$

(where, as usual $\cdot$ denotes product of $\mathbb{K}$).

We call a stream $\pi \in \mathbb{K}^\omega$ *polynomial* if there are $n \geq 0$ and $a_i \in \mathbb{K}$ such that

$$\pi = a_0 + a_1\mathcal{X} + a_2\mathcal{X}^2 + \cdots + a_n\mathcal{X}^n \ = \ (a_0,\, a_1,\, a_2,\, \ldots,\, a_n,\, 0,\, 0,\, 0,\, \ldots)$$

where we write $a_i\mathcal{X}^i$ for $[a_i] \times \mathcal{X}^i$ with $\mathcal{X}^i$ the i-fold product of $\mathcal{X}$ with itself.

A stream σ with $\sigma(0) \neq 0$ has a (unique) multiplicative inverse σ^{-1} in $\mathbb{K}^\omega$, satisfying

$$\sigma^{-1} \times \sigma = [1]$$

As usual, we shall often write $1/\sigma$ for σ^{-1} and σ/τ for $\sigma \times \tau^{-1}$. Note that the initial value of the sum, product and inverse of streams is given by the sum, product and inverse of their initial values.

We call a stream $\rho \in \mathbb{K}^\omega$ *rational* if it is the quotient $\rho = \sigma/\tau$ of two polynomial streams σ and τ with $\tau(0) \neq 0$.

One can compute a stream from its initial value and derivative by the so-called *fundamental theorem* of stream calculus [29]: for all $\sigma \in \mathbb{K}^\omega$,

$$\sigma = \sigma(0) + (\mathcal{X} \times \sigma') \tag{4}$$

(writing $\sigma(0)$ for $[\sigma(0)]$).

The fundamental theorem of stream calculus allows us to solve *stream differential equations* such as

$$\sigma' = 3 \times \sigma \ , \quad \sigma(0) = 1$$

by computing $\sigma = \sigma(0) + (\mathcal{X} \times \sigma') = 1 + (\mathcal{X} \times 3 \times \sigma)$, which leads to the solution

$$\sigma = 1/(1 - 3\mathcal{X}) = (1,\, 3,\, 3^2,\, 3^3,\, \ldots)$$

5.2. Solving linear systems of stream differential equations

Using some elementary linear algebra notation (matrices and vectors), we next show how to solve *linear* systems of stream differential equations. For notational convenience, we shall deal with linear systems of dimension 2, which can be straightforwardly generalised to systems of higher dimensions. They are given by the following data:

$$\begin{pmatrix}\sigma\\ \tau\end{pmatrix}' = M \times \begin{pmatrix}\sigma\\ \tau\end{pmatrix} \qquad \begin{pmatrix}\sigma\\ \tau\end{pmatrix}(0) = N \tag{5}$$

where M is a 2×2-matrix and N is a 1×2-matrix over $\mathbb{K}$:

$$M = \begin{pmatrix}m_{11} & m_{12}\\ m_{21} & m_{22}\end{pmatrix} \qquad N = \begin{pmatrix}n_1\\ n_2\end{pmatrix}$$

for $m_{ij},\ n_i \in \mathbb{K}$. The above notation is really just a short hand for the following system of two stream differential equations:

$$\begin{aligned} \sigma' &= (m_{11} \times \sigma) + (m_{12} \times \tau) \qquad & \sigma(0) = n_1 \\ \tau' &= (m_{21} \times \sigma) + (m_{22} \times \tau) \qquad & \tau(0) = n_2 \end{aligned}$$

We can solve such a system of equations by using twice the fundamental theorem of stream calculus (equation (4) above), once for σ and once for τ:

$$\begin{aligned} \sigma &= \sigma(0) + (\mathcal{X} \times \sigma') \\ \tau &= \tau(0) + (\mathcal{X} \times \tau') \end{aligned}$$

In matrix notation, the fundamental theorem looks like

$$\begin{pmatrix}\sigma\\ \tau\end{pmatrix} = \begin{pmatrix}\sigma\\ \tau\end{pmatrix}(0) + \mathcal{X} \times \begin{pmatrix}\sigma\\ \tau\end{pmatrix}'$$

Next we can solve our linear system (5) above by happily calculating as follows:

$$\begin{aligned} \begin{pmatrix}\sigma\\ \tau\end{pmatrix} &= \begin{pmatrix}\sigma\\ \tau\end{pmatrix}(0) + \mathcal{X} \times \begin{pmatrix}\sigma\\ \tau\end{pmatrix}' \\ &= N + \mathcal{X} \times M \times \begin{pmatrix}\sigma\\ \tau\end{pmatrix} \end{aligned}$$

This leads to

$$(I - (\mathcal{X} \times M)) \begin{pmatrix}\sigma\\ \tau\end{pmatrix} = N$$

where I and $\mathcal{X} \times M$ are given by

$$I = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \qquad \mathcal{X} \times M = \begin{pmatrix}m_{11} \times \mathcal{X} & m_{12} \times \mathcal{X}\\ m_{21} \times \mathcal{X} & m_{22} \times \mathcal{X}\end{pmatrix}$$

Finally, we can express the unique solution of our linear system of stream differential equations as follows:

$$\begin{pmatrix}\sigma\\ \tau\end{pmatrix} = (I - (\mathcal{X} \times M))^{-1} \times N$$

The advantage of the matrix notations above now becomes clear: we can compute the inverse of the matrix

$$(I - (\mathcal{X} \times M)) = \begin{pmatrix}1 - (m_{11} \times \mathcal{X}) & -(m_{12} \times \mathcal{X})\\ -(m_{21} \times \mathcal{X}) & 1 - (m_{22} \times \mathcal{X})\end{pmatrix}$$

whose values are simple polynomial streams, by standard linear algebra.

Let us look at an example. For

$$M = \begin{pmatrix} 0 & 1 \\ -1 & 2 \end{pmatrix} \qquad N = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

our linear system of stream differential equations (5) has the following solution:

$$\begin{aligned} \begin{pmatrix} \sigma \\ \tau \end{pmatrix} &= (I - (\mathcal{X} \times M))^{-1} \times N \\ &= \begin{pmatrix} 1 & -\mathcal{X} \\ \mathcal{X} & 1 - 2\mathcal{X} \end{pmatrix}^{-1} \times \begin{pmatrix} 1 \\ 2 \end{pmatrix} \\ &= \begin{pmatrix} \frac{1-2\mathcal{X}}{(1-\mathcal{X})^2} & \frac{\mathcal{X}}{(1-\mathcal{X})^2} \\ \frac{-\mathcal{X}}{(1-\mathcal{X})^2} & \frac{1}{(1-\mathcal{X})^2} \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{(1-\mathcal{X})^2} \\ \frac{2-\mathcal{X}}{(1-\mathcal{X})^2} \end{pmatrix} \end{aligned}$$

We note that the solutions of linear systems of stream differential equations always consist of rational streams.

5.3. Characterising the final morphism: $|A| = 1$

It is easy to see that when $|A| = 1$, the final coalgebra for the functor $\mathcal{L}$ is $(\mathbb{K}^\omega, \langle (-)(0), (-)' \rangle)$ where $(-)(0){:}\mathbb{K}^\omega \to \mathbb{K}$ and $(-)'{:}\mathbb{K}^\omega \to \mathbb{K}^\omega$ map each stream σ in its initial value $\sigma(0)$ and in its stream derivative σ'. Let $(\mathbb{K}^2, \langle o, t \rangle)$ be a LWA, with linear maps $o{:}\, \mathbb{K}^2 \to \mathbb{K}$ and $t{:}\, \mathbb{K}^2 \to \mathbb{K}^2$ that are represented by a 1×2-matrix O and by a 2×2-matrix T. We will now show how the final homomorphism

$$\begin{array}{ccc} \mathbb{K}^2 & \xrightarrow{[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^2}} & \mathbb{K}^\omega \\ {\scriptstyle \langle o,t \rangle} \downarrow & & \downarrow {\scriptstyle \langle (-)(0), (-)' \rangle} \\ \mathbb{K} \times \mathbb{K}^2 & \xrightarrow[id_{\mathbb{K}} \times [\![-]\!]^{\mathcal{L}}_{\mathbb{K}^2}]{} & \mathbb{K} \times \mathbb{K}^\omega \end{array}$$

can be characterised in terms of rational streams. To this end, we define

$$\sigma = [\![\begin{pmatrix} 1 \\ 0 \end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2} \qquad \tau = [\![\begin{pmatrix} 0 \\ 1 \end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2}$$

It follows from the commutativity of the diagram above that

$$\sigma' = [\![(T \begin{pmatrix} 1 \\ 0 \end{pmatrix})]\!]^{\mathcal{L}}_{\mathbb{K}^2} \qquad \sigma(0) = O \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\tau' = [\![(T \begin{pmatrix} 0 \\ 1 \end{pmatrix})]\!]^{\mathcal{L}}_{\mathbb{K}^2} \qquad \tau(0) = O \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

and this can be concisely expressed by the following system:

$$\begin{pmatrix} \sigma \\ \tau \end{pmatrix}' = {}^{\mathrm{t}}T \times \begin{pmatrix} \sigma \\ \tau \end{pmatrix} \qquad \begin{pmatrix} \sigma \\ \tau \end{pmatrix}(0) = {}^{\mathrm{t}}O$$

(where the superscript t indicates matrix transpose). These identities present σ and τ as the solution of a linear system of stream differential equations. By the results from Subsection 5.2, it follows that

$$\begin{pmatrix} \sigma \\ \tau \end{pmatrix} = (I - (\mathcal{X} \times {}^{\mathrm{t}}T))^{-1} \times {}^{\mathrm{t}}O$$

which leads to the following general formula for $[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^2}$:

$$[\![\begin{pmatrix}k_1\\k_2\end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2} = \begin{pmatrix}k_1 & k_2\end{pmatrix} \times (I - (\mathcal{X} \times {}^{t}T))^{-1} \times {}^{t}O$$

For instance, if

$$T = \begin{pmatrix}0 & -1\\1 & 2\end{pmatrix} \qquad O = \begin{pmatrix}1 & 2\end{pmatrix}$$

we find, using the example with M and N from Subsection 5.2, that

$$\begin{aligned}
[\![\begin{pmatrix}k_1\\k_2\end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2} &= \begin{pmatrix}k_1 & k_2\end{pmatrix} \times (I - (\mathcal{X} \times T^t))^{-1} \times O^t \\
&= \begin{pmatrix}k_1 & k_2\end{pmatrix} \times (I - (\mathcal{X} \times M))^{-1} \times N \\
&= \begin{pmatrix}k_1 & k_2\end{pmatrix} \times \begin{pmatrix}\frac{1}{(1-\mathcal{X})^2}\\ \frac{2-\mathcal{X}}{(1-\mathcal{X})^2}\end{pmatrix} \\
&= \frac{(k_1 + 2k_2) - k_2\mathcal{X}}{(1-\mathcal{X})^2}
\end{aligned}$$

Note that the above expression fully characterizes $[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^2}$, in the sense that it maps each $v \in \mathbb{K}^2$ in the corresponding rational stream.

Computing $\approx_{\mathcal{L}}$. We can employ the above characterization in order to compute $\approx_{\mathcal{L}}$ on $(\mathbb{K}^2, \langle o, t\rangle)$. We use the fact that the final homomorphism identifies precisely all equivalent states:

$$\begin{aligned}
\begin{pmatrix}x_1\\x_2\end{pmatrix} \approx_{\mathcal{L}} \begin{pmatrix}y_1\\y_2\end{pmatrix} &\iff [\![\begin{pmatrix}x_1\\x_2\end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2} = [\![\begin{pmatrix}y_1\\y_2\end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2} \\
&\iff [\![\begin{pmatrix}x_1 - y_1\\x_2 - y_2\end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2} = 0
\end{aligned}$$

where the 0 on the right is the stream $[0] = (0, 0, 0, \ldots)$. The kernel of the final homomorphism can now be computed using our characterisation above: for all $k_1, k_2 \in \mathbb{K}$,

$$\begin{aligned}
[\![\begin{pmatrix}k_1\\k_2\end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2} = 0 &\iff \frac{(k_1 + 2k_2) - k_2\mathcal{X}}{(1-\mathcal{X})^2} = 0 \\
&\iff (k_1 + 2k_2) - k_2\mathcal{X} = 0 \\
&\iff k_1 = 0 \text{ and } k_2 = 0
\end{aligned}$$

As a consequence, we find, for the present example:

$$\begin{pmatrix}x_1\\x_2\end{pmatrix} \approx_{\mathcal{L}} \begin{pmatrix}y_1\\y_2\end{pmatrix} \iff \begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}y_1\\y_2\end{pmatrix}$$

5.4. Rational weighted languages

All the results presented above allow to characterize the final homomorphism for weighted automata over an alphabet with a single letter. These results can be generalized in order to deal with alphabets of size greater than one.

Let A be an arbitrary finite alphabet. Recall from Section 3.3 that the final $\mathcal{L}$-coalgebra is $(\mathbb{K}^{A^*}, \langle \epsilon, d\rangle)$ where for all $\sigma \in \mathbb{K}^{A^*}$ and $a \in A$,

$$\epsilon(\sigma) = \sigma(\epsilon) \qquad d(\sigma)(a) = \sigma_a$$

and σ_a denotes the a-derivatives of the language σ.

The calculus presented in the previous section for one-variable power series (streams) can be generalized for multiple variable series [28], which we will recall next.

There are unique operators on series satisfying the following equations. For all $k \in \mathbb{K}$, $a, b \in A$ and $\sigma, \tau \in \mathbb{K}^{A^*}$,

Derivative	Initial Value	Name
$k_a = 0$	$k(\epsilon) = k$	Constant
$(\mathcal{X}_a)_a = 1, (\mathcal{X}_a)_b = 0 \ (b \neq a)$	$\mathcal{X}_a(\epsilon) = 0$	Variable
$(\sigma + \tau)_a = \sigma_a + \tau_a$	$(\sigma + \tau)(\epsilon) = \sigma(\epsilon) + \tau(\epsilon)$	Sum
$(\sigma \times \tau)_a = (\sigma_a \times \tau) + (\sigma(\epsilon) \times \tau_a)$	$(\sigma \times \tau)(\epsilon) = \sigma(\epsilon) \times \tau(\epsilon)$	Convolution product
$(\sigma^{-1})_a = -(\sigma(\epsilon)^{-1} \times \sigma_a) \times \sigma^{-1}$	$(\sigma^{-1})(\epsilon) = \sigma(\epsilon)^{-1}$, if $\sigma(\epsilon) \neq 0$	Inverse

A weighted language is *rational* if it can be constructed from finitely many constants $k \in \mathbb{K}$ and variables $\mathcal{X}_a$, by means of the operators of sum, product, and inverse. Rational languages constitute the class of languages that are recognized by finite dimensional weighted automata.

As for streams, one can compute a series from its initial value and derivatives by the so-called fundamental theorem [28]. That is, for all weighted languages $\sigma \in \mathbb{K}^{A^*}$:

$$\sigma = \sigma(\epsilon) + \sum_{a \in A} \mathcal{X}_a \times \sigma_a \tag{6}$$

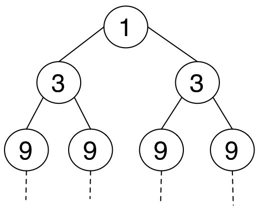
The fundamental theorem allows us to solve equations, similar to what happened above for streams. As an example, take $A = \{a, b\}$ (weighted languages over two symbols coincide with infinite binary trees), and the following equations

$$\sigma_a = 3 \times \sigma, \quad \sigma_b = 3 \times \sigma, \quad \sigma(\epsilon) = 1$$

Applying the fundamental theorem we reason as follows:

$$\begin{array}{lll} & \sigma & = \sigma(\epsilon) + (\mathcal{X}_a \times \sigma_a) + (\mathcal{X}_b \times \sigma_b) \\ \Leftrightarrow & \sigma & = 1 + (3\mathcal{X}_a \times \sigma) + (3\mathcal{X}_b \times \sigma) \\ \Leftrightarrow & (1 - 3\mathcal{X}_a - 3\mathcal{X}_b)\sigma & = 1 \end{array}$$

which leads to the solution $\sigma = (1 - 3\mathcal{X}_a - 3\mathcal{X}_b)^{-1}$, the tree depicted in the following picture.



Note that the above language is exactly the one recognized by the automata in Figure 1. It is also interesting to remark the strong similarity with streams: the formula for the stream $(1, 3, 6, 9, \ldots)$ is $(1 - 3\mathcal{X})^{-1}$.

Now that we know how to compute the solution of a single equation, moving to systems of equations is precisely as for streams. Again, for notational convenience, we shall exemplify with linear systems of dimension 2. The goal is to solve

$$\begin{pmatrix} \sigma \\ \tau \end{pmatrix}_a = M_a \times \begin{pmatrix} \sigma \\ \tau \end{pmatrix} \qquad \begin{pmatrix} \sigma \\ \tau \end{pmatrix}(\epsilon) = N$$

where, for each $a \in A$, M_a is a 2×2-matrix and N is a 1×2-matrix over $\mathbb{K}$.

We now solve this system by calculating as follows (similar to the stream case), now using the fundamental theorem for weighted languages, given in equation (6):

$$\begin{aligned} \begin{pmatrix} \sigma \\ \tau \end{pmatrix} &= \begin{pmatrix} \sigma \\ \tau \end{pmatrix}(\epsilon) + \sum_{a \in A} \mathcal{X}_a \times \begin{pmatrix} \sigma \\ \tau \end{pmatrix}_a \\ &= N + \sum_{a \in A} \mathcal{X}_a \times M_a \times \begin{pmatrix} \sigma \\ \tau \end{pmatrix} \end{aligned}$$

This leads to

$$\left(I - \sum_{a \in A} (\mathcal{X}_a \times M_a) \right) \begin{pmatrix} \sigma \\ \tau \end{pmatrix} \; = \; N$$

where I and $\mathcal{X}_a \times M_a$ are as before.

Finally, we can express the unique solution of our linear system as follows:

$$\begin{pmatrix} \sigma \\ \tau \end{pmatrix} \; = \; \left(I - \sum_{a \in A} (\mathcal{X}_a \times M_a) \right)^{-1} \times \; N$$

Hence, the only difference with the stream case is that instead of computing the inverse of the matrix $I - (\mathcal{X} \times M)$ one needs to compute the inverse of $I - \sum\limits_{a \in A} (\mathcal{X}_a \times M)$.

Some remarks on computing the inverse of $I - \sum\limits_{a \in A} (\mathcal{X}_a \times M)$ are now in order. Convolution product on power series in not commutative as soon as A has more than one element (e.g., $\mathcal{X}_a \times \mathcal{X}_b \neq \mathcal{X}_b \times \mathcal{X}_a$). Thus, the matrix above is a matrix with entries stemming from a non-commutative ring. Traditional methods (Gaussian elimination, Cramer's rule, ...) to compute the inverse of matrices are not applicable and thus one needs to resort to other (more complicated) techniques such as quasi-determinants [11] or generalized LDU decomposition [8].

A function to compute the inverse of a matrix with non-commutative entries is provided in the *Mathematica* [22] package `NCAlgebra` [25]. The algorithm implemented in the package is directly base in LDU decomposition [8]. The matrices we show below were all obtained using the aforementioned package.

For instance, for $A = \{a, b, c\}$, if

$$M_a = M_c = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix} \qquad M_b = \begin{pmatrix} 0 & 0.5 \\ 0 & 0.5 \end{pmatrix} \qquad N = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

then

$$I - \mathcal{X}_a \times M_a - \mathcal{X}_b \times M_b - \mathcal{X}_c \times M_c = \begin{pmatrix} 1 - 2\mathcal{X}_a - 2\mathcal{X}_c & -0.5\mathcal{X}_b \\ 0 & 1 - 0.5\mathcal{X}_b \end{pmatrix}$$

and

$$(I - \mathcal{X}_a \times M_a - \mathcal{X}_b \times M_b - \mathcal{X}_c \times M_c)^{-1} = \begin{pmatrix} \frac{1}{1-2\mathcal{X}_a-2\mathcal{X}_c} & 0.5\frac{1}{1-2\mathcal{X}_a-2\mathcal{X}_c}\mathcal{X}_b\frac{1}{1-0.5\mathcal{X}_b} \\ 0 & 1 - 0.5\mathcal{X}_b \end{pmatrix}$$

The final homomorphism $[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^2}$ is represented in the following diagram

$$\begin{array}{ccc} \mathbb{K}^2 & \xrightarrow{[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^2}} & \mathbb{K}^{A^*} \\ \big\downarrow{\scriptstyle \langle o,t \rangle} & & \big\downarrow{\scriptstyle \langle \epsilon,d \rangle} \\ \mathbb{K} \times \mathbb{K}^{2^A} & \xrightarrow[id_{\mathbb{K}} \times [\![-]\!]^{\mathcal{L}}_{\mathbb{K}^{2^A}}]{} & \mathbb{K} \times \mathbb{K}^{A^*} \end{array}$$

where, as usual, o and $t = \{t_a{:}\mathbb{K}^2 \to \mathbb{K}^2\}_{a \in A}$ are linear mappings represented by the 1×2-row vector O and the 2×2-matrixes T_a, respectively.

We will show how the final homomorphism $[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^2}$ can be characterized in terms of rational weighted languages. To this end, we again define

$$\sigma = [\![\begin{pmatrix} 1 \\ 0 \end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2} \qquad \tau = [\![\begin{pmatrix} 0 \\ 1 \end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2}$$

It follows from the commutativity of the diagram above that

$$\sigma_a = [\![(T_a \begin{pmatrix} 1 \\ 0 \end{pmatrix})]\!]^{\mathcal{L}}_{\mathbb{K}^2} \qquad \sigma(\epsilon) = O \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\tau_a = [\![(T_a \begin{pmatrix} 0 \\ 1 \end{pmatrix})]\!]^{\mathcal{L}}_{\mathbb{K}^2} \qquad \tau(\epsilon) = O \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

and this can be concisely expressed by the following system:

$$\begin{pmatrix} \sigma \\ \tau \end{pmatrix}_a = {}^{\mathrm{t}}T_a \times \begin{pmatrix} \sigma \\ \tau \end{pmatrix} \qquad \begin{pmatrix} \sigma \\ \tau \end{pmatrix}(\epsilon) = {}^{\mathrm{t}}O$$

It then follows that

$$\begin{pmatrix} \sigma \\ \tau \end{pmatrix} = (I - \left(\sum_{a\in A} \mathcal{X}_a \times {}^{\mathrm{t}}T_a\right))^{-1} \times {}^{\mathrm{t}}O$$

which leads to the following general formula for $[\![-]\!]^{\mathcal{L}}_{\mathbb{K}^2}$:

$$[\![\begin{pmatrix} k_1 \\ k_2 \end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2} = \begin{pmatrix} k_1 & k_2 \end{pmatrix} \times (I - \left(\sum_{a\in A} \mathcal{X}_a \times {}^{\mathrm{t}}T_a\right))^{-1} \times {}^{\mathrm{t}}O$$

For instance, for $A = \{a, b, c\}$ and

$$T_a = T_c = \begin{pmatrix} 2 & 0 \\ 0 & 0 \end{pmatrix} \qquad T_b = \begin{pmatrix} 0 & 0 \\ 0.5 & 0.5 \end{pmatrix} \qquad O = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

we find, using the example above, that

$$\begin{aligned}
[\![\begin{pmatrix} k_1 \\ k_2 \end{pmatrix}]\!]^{\mathcal{L}}_{\mathbb{K}^2} &= \begin{pmatrix} k_1 & k_2 \end{pmatrix} \times \left(\sum_{a\in A} \mathcal{X}_a \times {T_a}^t\right)^{-1} \times O^t \\
&= \begin{pmatrix} k_1 & k_2 \end{pmatrix} \times \left(\sum_{a\in A} \mathcal{X}_a \times T_a\right)^{-1} \times N \\
&= \begin{pmatrix} k_1 & k_2 \end{pmatrix} \times \begin{pmatrix} \frac{1}{1-2\mathcal{X}_a-2\mathcal{X}_c} + 0.5\frac{1}{1-2\mathcal{X}_a-2\mathcal{X}_c}\mathcal{X}_b\frac{1}{1-0.5\mathcal{X}_b} \\ \frac{1}{(1-0.5\mathcal{X}_b)} \end{pmatrix} \\
&= \frac{k_1}{1-2\mathcal{X}_a-2\mathcal{X}_c} + 0.5k_1\frac{1}{1-2\mathcal{X}_a-2\mathcal{X}_c}\mathcal{X}_b\frac{1}{1-0.5\mathcal{X}_b} + \frac{k_2}{(1-0.5\mathcal{X}_b)}
\end{aligned}$$

By generalizing the above arguments from $\mathbb{K}^2$ to any finite dimesion vector space, we obtain the following theorem.

Theorem 5 *Let $(V, \langle o, t\rangle)$ be a linear weighted automata with V finite dimension. Then, for all $v \in V$*

$$[\![v]\!]^{\mathcal{L}}_V = {}^{\mathrm{t}}v \times (I - \left(\sum_{a\in A} \mathcal{X}_a \times {}^{\mathrm{t}}T_a\right))^{-1} \times {}^{\mathrm{t}}O$$

For an example with a three dimensional state space, we consider the LWA corresponding to the automaton $(V, \langle o, t\rangle)$ in Fig. 5.

$$\llbracket \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix} \rrbracket_V^{\mathcal{L}} = \begin{pmatrix} k_1 & k_2 & k_3 \end{pmatrix} \times (I - \left(\sum_{a \in A} \mathcal{X}_a \times {}^{t}T_a \right))^{-1} \times {}^{t}O$$

$$= \begin{pmatrix} k_1 & k_2 & k_3 \end{pmatrix} \times (I - \begin{pmatrix} \mathcal{X}_a + \mathcal{X}_b & 0 & 0 \\ \frac{\mathcal{X}_a}{3} & 0 & \frac{\mathcal{X}_b}{3} \\ \mathcal{X}_a & 3\mathcal{X}_b & 0 \end{pmatrix})^{-1} \times \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} k_1 & k_2 & k_3 \end{pmatrix} \times \begin{pmatrix} 1 - \mathcal{X}_a - \mathcal{X}_b & 0 & 0 \\ -\frac{\mathcal{X}_a}{3} & 1 & -\frac{\mathcal{X}_b}{3} \\ -\mathcal{X}_a & -3\mathcal{X}_b & 1 \end{pmatrix}^{-1} \times \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

The inverse of the matrix in the middle is

$$M = \begin{pmatrix} \frac{1}{1-\mathcal{X}_a-\mathcal{X}_b} & 0 & 0 \\ (\frac{1}{3} + \frac{\mathcal{X}_b}{3}\frac{1}{1-\mathcal{X}_b^2}(\mathcal{X}_b+1))\mathcal{X}_a \frac{1}{1-\mathcal{X}_a-\mathcal{X}_b} & 1 + \mathcal{X}_b \frac{1}{1-\mathcal{X}_b^2}\mathcal{X}_b & \frac{\mathcal{X}_b}{3}\frac{1}{1-\mathcal{X}_b^2} \\ (\frac{1}{1-\mathcal{X}_b^2})(\mathcal{X}_a + \mathcal{X}_b\mathcal{X}_a)\frac{1}{1-\mathcal{X}_a-\mathcal{X}_b} & 3\frac{1}{1-\mathcal{X}_b^2}\mathcal{X}_b & \frac{1}{1-\mathcal{X}_b^2} \end{pmatrix}$$

and

$$M \times \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} \\ (\frac{1}{3} + \frac{\mathcal{X}_b}{3}\frac{1}{1-\mathcal{X}_b^2}(\mathcal{X}_b+1))\mathcal{X}_a \frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} + 1 + \mathcal{X}_b\frac{1}{1-\mathcal{X}_b^2}\mathcal{X}_b + \frac{\mathcal{X}_b}{3}\frac{1}{1-\mathcal{X}_b^2} \\ (\frac{1}{1-\mathcal{X}_b^2})(\mathcal{X}_a + \mathcal{X}_b\mathcal{X}_a)\frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} + 3\frac{1}{1-\mathcal{X}_b^2}\mathcal{X}_b + \frac{1}{1-\mathcal{X}_b^2} \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \end{pmatrix}$$

Summarizing

$$\llbracket \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix} \rrbracket_V^{\mathcal{L}} = \begin{pmatrix} k_1 & k_2 & k_3 \end{pmatrix} \times \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \end{pmatrix} \tag{7}$$

Note that the above expression fully characterizes $\llbracket - \rrbracket_V^{\mathcal{L}}$, in the sense that it maps each $v \in V$ in the rational weighted lanuage that it accepts.

Computing $\approx_{\mathcal{L}}$. Now, we have a rational expression $\sigma = k_1\rho_1 + k_2\rho_2 + k_3\rho_3$ characterizing the final homomorphism and we would like to calculate for which values of k_1, k_2 and k_3 this expression equals 0. As we have shown before, when $|A| = 1$, this can be done by syntactically manipulating the rational expression in a standard way. In the general case, because of the non commutativity of the convolution product, this is not trivial at all.

Here, we choose to adopt the following approach: first we compute "some" derivatives σ_a, σ_b, σ_{aa}, σ_{ab} ... and then we check for wich k_1, k_2 and k_3 the initial values $\sigma(\epsilon)$, $\sigma_a(\epsilon)$, $\sigma_b(\epsilon)$, $\sigma_{aa}(\epsilon)$, $\sigma_{ab}(\epsilon)$ ... are equal to 0. The following lemma (proved in [5, 28]) ensures that we have to compute only finitely many derivatives.

Lemma 5 *Rational weighted languages have finitely many linearly independent derivatives.*

In our example, we start by taking the initial value of the expression σ itself obtaining $\sigma(\epsilon) = 2k_1 + k_2 + k_3$. Then we take the a and b derivatives which give, respectively, the expressions

$$\sigma_a = k_1(\rho_1)_a + k_2(\rho_2)_a + k_3(\rho_3)_a \tag{8}$$

$$\begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \end{pmatrix}_a = \begin{pmatrix} \frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} \\ \frac{1}{3}\frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} \\ \frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} \end{pmatrix}$$

and

$$\sigma_b = k_1(\rho_1)_b + k_2(\rho_2)_b + k_3(\rho_3)_b$$

$$\begin{pmatrix}\rho_1\\ \\ \rho_2\\ \\ \rho_3\end{pmatrix}_b = \begin{pmatrix} \frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} \\ (\frac{1}{3}\frac{1}{1-\mathcal{X}_b^2}(\mathcal{X}_b+1))\mathcal{X}_a\frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} + \frac{1}{1-\mathcal{X}_b^2}\mathcal{X}_b + \frac{1}{3}\frac{1}{1-\mathcal{X}_b^2} \\ \mathcal{X}_b(\frac{1}{1-\mathcal{X}_b^2})(\mathcal{X}_a+\mathcal{X}_b\mathcal{X}_a)\frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} + \mathcal{X}_a\frac{1}{1-\mathcal{X}_a-\mathcal{X}_b} + 3\mathcal{X}_b\frac{1}{1-\mathcal{X}_b^2}\mathcal{X}_b + 3 + \mathcal{X}_b\frac{1}{1-\mathcal{X}_b^2} \end{pmatrix}$$

which have initial values $\sigma_a(\epsilon) = 2k_1 + \frac{2}{3}k_2 + 2k_3$ and $\sigma_b(\epsilon) = 2k_1 + \frac{1}{3}k_2 + 3k_3$.

Now, note that the a derivative, that is the rational expression (8), will now always generate the same derivatives for a and b (since the derivatives of $\frac{2}{1-\mathcal{X}_a-\mathcal{X}_b}$ are the expression itself again; intuitively, this expression represents an infinite binary tree with 2's in every node and hence has left and right subtrees equal to the whole tree). For the b derivative, we take another level of derivatives and obtain, respectively,

$$\sigma_{ba} = k_1(\rho_1)_{ba} + k_2(\rho_2)_{ba} + k_3(\rho_3)_{ba}$$

$$\begin{pmatrix}\rho_1\\ \rho_2\\ \rho_3\end{pmatrix}_{ba} = \begin{pmatrix} \frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} \\ \frac{1}{3}\frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} \\ \frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} \end{pmatrix} = \begin{pmatrix}\rho_1\\ \rho_2\\ \rho_3\end{pmatrix}_a$$

and

$$\sigma_{bb} = k_1(\rho_1)_{bb} + k_2(\rho_2)_{bb} + k_3(\rho_3)_{bb}$$

$$\begin{pmatrix}\rho_1\\ \\ \rho_2\\ \\ \rho_3\end{pmatrix}_{bb} = \begin{pmatrix} \frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} \\ (\frac{1}{3}\mathcal{X}_b\frac{1}{1-\mathcal{X}_b^2}(\mathcal{X}_b+1) + \frac{1}{3})\mathcal{X}_a\frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} + \mathcal{X}_b\frac{1}{1-\mathcal{X}_b^2}\mathcal{X}_b + 1 + \frac{1}{3}\mathcal{X}_b\frac{1}{1-\mathcal{X}_b^2} \\ (\frac{1}{1-\mathcal{X}_b^2})(\mathcal{X}_a+\mathcal{X}_b\mathcal{X}_a)\frac{2}{1-\mathcal{X}_a-\mathcal{X}_b} + 3\frac{1}{1-\mathcal{X}_b^2}\mathcal{X}_b + \frac{1}{1-\mathcal{X}_b^2} \end{pmatrix} = \begin{pmatrix}\rho_1\\ \\ \rho_2\\ \\ \rho_3\end{pmatrix}$$

The a-derivative coincides with (8) and the b derivative coincides with the original expression σ. Therefore, we have found the the system of equations we need to solve:

$$\begin{cases} \sigma(\epsilon) = 0 \\ \sigma_a(\epsilon) = 0 \\ \sigma_b(\epsilon) = 0 \end{cases} \Leftrightarrow \begin{cases} 2k_1 + k_2 + k_3 = 0 \\ 2k_1 + \frac{2}{3}k_2 + 2k_3 = 0 \\ 2k_1 + \frac{1}{3}k_2 + 3k_3 = 0 \end{cases}$$

Solving it yields $k_1 = -2k_3$ and $k_2 = 3k_3$. Hence, the kernel of the final homomorphism is the space spanned by the vector

$$\begin{pmatrix} -2 \\ 3 \\ 1 \end{pmatrix}$$

which coincides with what was computed by the forward algorithm in Section 4.1.

This example also shows that this procedure is in general not more efficient then the forward algorithm. Indeed, the three equations of the above system coincide with the spaces computed by the forward algorithm: the space (of solutions) of $\sigma(\epsilon) = 0$ is the space spanned by B_0 (in Section 4.1), the space of $\sigma_a(\epsilon) = 0$ is the one spanned by B_1^a, and the space $\sigma_a(\epsilon) = 0$ is the one spanned by B_2^a.

6. Discussion

In this paper we proposed a novel coalgebraic perspective on weighted automata and their behavioural equivalences. Weighted automata are $\mathcal{W}$-coalgebras, for a functor $\mathcal{W}$ on Set, but they can be regarded also as linear weighted automata, that are $\mathcal{L}$-coalgebras for a functor $\mathcal{L}$ on $Vect$. The behavioural equivalence induced by $\mathcal{W}$ coincides with weighted bisimilarity, while the equivalence induced by $\mathcal{L}$ ($\approx_{\mathcal{L}}$) with weighted language equivalence.

Weighted languages (i.e. formal power series) form the vectors spaces $\mathbb{K}^{A^*}$ that carries the final $\mathcal{L}$-coalgebra: for each linear weighted automata $(V, \langle o, t \rangle)$, the unique $\mathcal{L}$-morphism $[\![-]\!]_V^{\mathcal{L}}$ into the final coalgebra maps each vector $v \in V$ into the weighted language in $\mathbb{K}^{A^*}$ that v accepts. The unique morphism $[\![-]\!]_V^{\mathcal{L}}$ is a linear map and its kernel coincides with $\approx_{\mathcal{L}}$ that, when V is finite dimension, can be computed in three different ways. It is important to remark here that the linearity of $[\![-]\!]_V^{\mathcal{L}}$ is key ingredient (in all the three approaches) to finitely compute the equivalence on an infinite state space.

Theorem 5 provides an explicit characterization of $[\![-]\!]_V^{\mathcal{L}}$ by assigning a syntactic expression denoting a rational weighted language to each vector $v \in V$. This characterization can be employed for computing $\approx_{\mathcal{L}}$ but, in general terms, it seems to be inconvenient to be implemented in an automatic prover. The backward algorithm, instead, is very efficient but its presentation is a bit complex since it requires dual spaces and transpose maps. The forward algorithm is easier to explain and we have shown it is closely related to the construction of the final coalgebra.

As a future work, we would like to extend these results to automata with weights on a semiring $\mathbb{S}$ (instead of field $\mathbb{K}$). The coalgebraic characterization of weighted bisimilarity can be easily obtained by employing a *semiring evaluation functor* instead of the field evaluation functor (Definition 2). For weighted language equivalence on semirings, we should define the functor $\mathcal{L}$ on the category of *semimodules*, instead of $Vect$. The forward algorithm could be extended (by exploiting its relationship with the construction of final coalgebras) in a rather straightforward way, but the convergence in a finite numbers of iterations might be not guaranteed. The other two approaches strongly rely on the properties of fields and vector spaces (such as the existence of the inverse multiplicative or the dual space). Therefore, it seems challenging to extend them to the case of a generic semiring $\mathbb{S}$. If $\mathbb{S}$ is a semifield however, then all elements have a multiplicative inverse. An important example of semifield in this context is the tropical semiring [15]. Further, when $\mathbb{S}$ is a commutative ring, annihilators and transpose maps can be generalized as operations carried out within the dual module (i.e. linear maps from an $\mathbb{S}$-module to $\mathbb{S}$, seen as a module) [26]. We leave these extensions as future work.

References

[1] Jirí Adámek, Horst Herrlich, and George E. Strecker. *Abstract and Concrete Categories - The Joy of Cats*. Wiley, 1990.

[2] Jürgen Albert and Jarkko Kari. *Handbook of Weighted Automata*, chapter Digital Image Compression, pages 213–250. Monographs in Theoretical Computer Science. Springer, 2009.

[3] Christel Baier, Marcus Größer, and Frank Ciesinski. *Handbook of Weighted Automata*, chapter Model Checking Linear-Time Properties of Probabilistic Systems, pages 213–250. Monographs in Theoretical Computer Science. Springer, 2009.

[4] Michael Barr. Terminal coalgebras in well-founded set theory. *Theor. Comput. Sci.*, 114(2):299–315, 1993.

[5] Jean Berstel and Christophe Reutenauer. *Rational Series and Their Languages*. Springer-Verlag, 1988.

[6] Michele Boreale. Weighted bisimulation in linear algebraic form. In *In Proc. of International Conference on the Theory of Concurrency (CONCUR), 2009*, volume 5710 of *Lecture Notes in Computer Science*, pages 163–177, 2009.

[7] Peter Buchholz. Bisimulation relations for weighted automata. *Theor. Comput. Sci.*, 393(1-3):109–123, 2008.

[8] Juan Francisco Camino, J. William Helton, and Robert E. Skelton. A symbolic algorithm for determining convexity of a matrix function: How to get schur complements out of your life. In *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000.

[9] Manfred Droste and Paul Gastin. Weighted automata and weighted logics. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 2005.

[10] Peter Freyd. *Abelian categories*. Harper and Row, 1964.

[11] Israel Gelfand, Sergei Gelfand, Vladimir Retakh, and Robert Lee Wilson. Quasideterminants. *Advances in Mathematics*, 193(1):56 – 141, 2005.

[12] H. Peter Gumm. Copower functors. *Theor. Comput. Sci.*, 410(12-13):1129–1142, 2009.

[13] H. Peter Gumm and Tobias Schröder. Monoid-labeled transition systems. *Electr. Notes Theor. Comput. Sci.*, 44(1), 2001.

[14] Paul Halmos. *Finite dimensional vector spaces*. Springer, 1974.

[15] Udo Hebisch and Hans Joachim Weinert. Semirings and semifields. In M. Hazewinkel, editor, *Handbook of Algebra*, volume 1, pages 425 – 462. North-Holland, 1996.

[16] Alberto Isidori. *Nonlinear Control Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 1995.

[17] Chi-Chang Jou and Scott A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In Jos C. M. Baeten and Jan Willem Klop, editors, *CONCUR*, volume 458 of *Lecture Notes in Computer Science*, pages 367–383. Springer, 1990.

[18] Paris C. Kanellakis and Scott A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990.

[19] Daniel Kirsten and Ina Mäurer. On the determinization of weighted automata. *Journal of Automata, Languages and Combinatorics*, 10(2/3):287–312, 2005.

[20] Werner Kuich. *Handbook of Formal Languages, Vol. 1, Word, Language, Grammar*, chapter Semirings and formal power series, page 609677. Springer-Verlag, 1997.
[21] Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.
[22] *Mathematica*. `http://www.wolfram.com/mathematica/`.
[23] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.
[24] Mehryar Mohri. *Handbook of Weighted Automata*, chapter Weighted Automata Algorithms, pages 213–250. Monographs in Theoretical Computer Science. Springer, 2009.
[25] The `NCAlgebra` package. `http://math.ucsd.edu/ñcalg/`.
[26] Joseph Rotman. *Advanced Modern Algebra*. Prentice-Hall, 2002.
[27] Jan J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.
[28] Jan J.M.M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comput. Sci.*, 308(1-3):1–53, 2003.
[29] Jan J.M.M. Rutten. A coinductive calculus of streams. *Mathematical Structures in Computer Science*, 15(1):93–147, 2005.
[30] Jan J.M.M. Rutten. Rational streams coalgebraically. *CoRR*, abs/0807.4073, 2008.
[31] Yousef Saad. *Iterative Methods for Sparse Linear Systems, Second Edition*. SIAM, 2003.
[32] Aarto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Texts and Monographs on Computer Science. Springer-Verlag, 1978.
[33] Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.
[34] Alexandra Silva. *Kleene Coalgebra*. PhD thesis, Radboud Universiteit Nijmegen, 2010.
[35] Eugene W. Stark. On behaviour equivalence for probabilistic i/o automata and its relationship to probabilistic bisimulation. *Journal of Automata, Languages and Combinatorics*, 8(2):361–395, 2003.