Algorithms for Weighted Multi-Tape Automata

– XRCE Research Report 2004/031 –

Andre Kempe[1] Franck Guingne[1,2] Florent Nicart[1,2]

[1] Xerox Research Centre Europe – Grenoble Laboratory
6 chemin de Maupertuis – 38240 Meylan – France
firstname.lastname@xrce.xerox.com – http://www.xrce.xerox.com

[2] Laboratoire d'Informatique Fondamentale et Appliquée de Rouen
Faculté des Sciences et des Techniques – Université de Rouen
76821 Mont-Saint-Aignan – France
firstname.lastname@dir.univ-rouen.fr – http://www.univ-rouen.fr/LIFAR/

June 2, 2004

Abstract

This report defines various operations and describes algorithms for *weighted multi-tape automata* (WMTAs). It presents, among others, a new approach to *multi-tape intersection*, meaning the intersection of a number of tapes of one WMTA with the same number of tapes of another WMTA, which can be seen as a generalization of transducer intersection. In our approach, multi-tape intersection is not considered as an atomic operation but rather as a sequence of more elementary ones. We show an example of multi-tape intersection, actually transducer intersection, that can be compiled with our approach but not with several other methods that we analyzed. Finally we describe an example of practical application, namely the preservation of intermediate results in transduction cascades.

Contents

1 Introduction

Finite state automata (FSAs) and weighted finite state automata (WFSAs) are well known, mathematically well defined, and offer many practical advantages. (Elgot and Mezei, 1965; Eilenberg, 1974; Kuich and Salomaa, 1986). They permit, among others, the fast processing of input strings and can be easily modified and combined by well defined operations. Both FSAs and WFSAs are widely used in language and speech processing (Kaplan and Kay, 1981; Koskenniemi, Tapanainen, and Voutilainen, 1992; Sproat, 1992; Karttunen et al., 1997; Mohri, 1997; Roche and Schabes, 1997). A number of software systems have been designed to manipulate FSAs and WFSAs (Karttunen et al., 1997; van Noord, 1997; Mohri, Pereira, and Riley, 1998; Beesley and Karttunen, 2003). Most systems and applications deal, however, only with *1-tape* and *2-tape automata*, also called acceptors and transducers, respectively.

Multi-tape automata (MTAs) (Elgot and Mezei, 1965; Kaplan and Kay, 1994) offer additional advantages such as the possibility of storing different types of information, used in NLP, on different tapes or preserving intermediate results of transduction cascades on different tapes so that they can be re-accessed by any of the following transductions. MTAs have been implemented and used, for example, in the morphological analysis of Semitic languages, where the vowels, consonants, pattern, and surface form of words have been represented on different tapes of an MTA (Kay, 1987; Kiraz, 1997; Kiraz and Grimley-Evans, 1998).

This report defines various operations for *weighted multi-tape automata* (WMTAs) and describes algorithms that have been implemented for those operations in the WFSC toolkit (Kempe et al., 2003). Some algorithms are new, others are known or similar to known algorithms. The latter will be recalled to make this report more complete and self-standing. We present a new approach to *multi-tape intersection*, meaning the intersection of a number of tapes of one WMTA with the same number of tapes of another WMTA. In our approach, multi-tape intersection is not considered as an atomic operation but rather as a sequence of more elementary ones, which facilitates its implementation. We show an example of multi-tape intersection, actually transducer intersection, that can be compiled with our approach but not with several other methods that we analyzed. To show the practical relevance of our work, we include an example of application: the preservation of intermediate results in transduction cascades.

For the structure of this report see the table of contents.

2 Some Previous Work

2.1 n-Tape Automaton Seen as a Two-Tape Automaton

Rabin and Scott (1959) presented in a survey paper a number of results and problems on finite 1-way automata, the last of which – the decidability of the equivalence of deterministic k-tape automata – has been solved only recently and by means of purely algebraic methods (Harju and Karhumäki, 1991).

Rabin and Scott considered the case of two-tape automata claiming this is not a loss of generality. They adopted the convention "... that the machine will read for a while on one tape, then change control and read a while on the other tape, and so on until one of the tapes is exhausted ...". In this view, a two-tape or n-tape machine is just an ordinary automaton with a partition of its states to determine which tape is to be read.

2.2 n-Tape Automaton Seen as a Single-Tape Automaton

Ganchev, Mihov, and Schulz (2003) define the notion of "one-letter k-tape automaton" and the main idea is to consider this restricted form of k-tape automata where all transition labels have exactly one tape with a non-empty single letter. Then they prove that one can use "classical" algorithms for 1-tape automata on a one-letter k-tape automaton. They propose an additional condition to be able to use classical intersection. It is based on the notion that a tape or coordinate is *inessential* iff $\forall\langle w_1, ..., w_k\rangle \in R$ (R is a regular relation over $(\Sigma^*)^k$) and $\forall v \in \Sigma^*$, $\langle w_1, ...w_{i-1}, v, w_{i+1}, ..., w_k\rangle \in R$. And thus to perform an intersection, they assume that there exists at most one common essential tape between the two operands.

2.3 n-Tape Transducer

Kaplan and Kay (1994) define a non-deterministic n*-way finite-state transducer* that is similar to a classic transducer except that the transition function maps $Q \times \Sigma^\epsilon \times ... \times \Sigma^\epsilon$ to 2^Q (with $\Sigma^\varepsilon = \Sigma \cup \{\varepsilon\}$). To perform the *intersection* between two n-tape transducers, they introduced the notion of *same-length relations* . As a result, they treat a subclass of n-tape transducers to be intersected.

Kiraz (1997) defines an n-tape finite state automaton and an n*-tape finite-state transducer*, introducing the notion of *domain tape* and *range tape* to be able to define a unambiguous composition for n-tape transducers. Operations on n-tape automata are based on (Kaplan and Kay, 1994) , the intersection in particular.

3 Mathematical Objects

In this section we recall the basic definitions of the algebraic structures monoid and semiring, and give a detailed definition of a weighted multi-tape automaton (WMTA) based on the definitions of a weighted automaton and a multi-tape automaton (Rabin and Scott, 1959; Elgot and Mezei, 1965; Eilenberg, 1974; Kuich and Salomaa, 1986).

3.1 Semirings

A *monoid* is a structure $\langle M, \circ, \bar{1}\rangle$ consisting of a set M, an associative binary operation $\circ$ on M, and a *neutral* element $\bar{1}$ such that $\bar{1} \circ a = a \circ \bar{1} = a$ for all $a \in M$. A monoid is called *commutative* iff $a \circ b = b \circ a$ for all $a, b \in M$.

A set $\mathbb{K}$ equipped with two binary operations, $\oplus$ (*collection*) and $\otimes$ (*extension*), and two neutral elements, $\bar{0}$ and $\bar{1}$, is called a *semiring*, iff it satisfies the following properties:

1. $\langle\mathbb{K}, \oplus, \bar{0}\rangle$ is a commutative monoid
2. $\langle\mathbb{K}, \otimes, \bar{1}\rangle$ is a monoid
3. extension is *left*- and *right-distributive* over collection:
   $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)\,, \quad (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)\,, \quad \forall a, b, c \in \mathbb{K}$
4. $\bar{0}$ is an annihilator for extension: $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}\,, \quad \forall a \in \mathbb{K}$

We denote a generic semiring as $\mathcal{K} = \langle\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1}\rangle$.

Some automaton algorithms require semirings to have specific properties. Composition, for example, requires it to be commutative (Pereira and Riley, 1997; Mohri, Pereira, and Riley, 1998) and ε-removal requires it to be *k-closed* (Mohri, 2002). These properties are defined as follows:

1. commutativity: $a \otimes b = b \otimes a \; , \;\; \forall a, b \in \mathbb{K}$
2. k-closedness: $\bigoplus_{n=0}^{k+1} a^n = \bigoplus_{n=0}^{k} a^n \; , \;\; \forall a \in \mathbb{K}$

The following well-known semirings are commutative:

1. $\mathcal{B} = \langle \mathbb{B}, \vee, \wedge, 0, 1 \rangle$: the boolean semiring, with $\mathbb{B} = \{0, 1\}$
2. $\mathcal{N} = \langle \mathbb{N}, +, \times, 0, 1 \rangle$: a positive integer semiring with arithmetic addition and multiplication
3. $\mathcal{R}^+ = \langle \mathbb{R}^+, +, \times, 0, 1 \rangle$: a positive real semiring
4. $\overline{\mathcal{R}}^+ = \langle \overline{\mathbb{R}}^+, min, +, \infty, 0 \rangle$: a real tropical semiring, with $\overline{\mathbb{R}}^+ = \mathbb{R}^+ \cup \{\infty\}$

A number of algorithms require semirings to be equipped with an order or partial order denoted by $<_\mathcal{K}$. Each idempotent semiring $\mathcal{K}$ (i.e., $\forall a \in \mathcal{K} : a \oplus a = a$) has a natural partial order defined by $a <_\mathcal{K} b \Leftrightarrow a \oplus b = a$. In the above examples, the boolean and the real tropical semiring are idempotent, and hence have a natural partial order.

3.2 Weighted Multi-Tape Automata

In analogy to a weighted automaton and a multi-tape automaton (MTA), we define a *weighted multi-tape automaton* (WMTA), also called weighted n-tape automaton, over a semiring $\mathcal{K}$, as a six-tuple

$$A^{(n)} \;=_{\text{def}}\; \langle \Sigma, Q, I, F, E^{(n)}, \mathcal{K} \rangle \tag{1}$$

with

Σ		being a finite alphabet
Q		the finite set of states
I	$\subseteq Q$	the set of initial states
F	$\subseteq Q$	the set of final states
n		the arity, i.e., the number of tapes of $A^{(n)}$
$E^{(n)}$	$\subseteq Q \times (\Sigma^*)^n \times \mathbb{K} \times Q$	being the finite set of n-tape transitions and
$\mathcal{K}$	$= \langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$	the semiring of weights.

For any state $q \in Q$,

$\lambda(q)$	$\in \mathcal{K}$	denotes its initial weight, with $\lambda(q) \neq \bar{0} \Leftrightarrow q \in I$,
$\varrho(q)$	$\in \mathcal{K}$	its final weight, with $\varrho(q) \neq \bar{0} \Leftrightarrow q \in F$, and
$E(q)$	$\subseteq E^{(n)}$	its finite set of out-going transitions.

For any transition $e^{(n)} \in E^{(n)}$, with $e^{(n)} = \langle p, \ell^{(n)}, w, n \rangle$,

$p(e^{(n)})$	$p : E^{(n)} \rightarrow Q$	denotes its source state
$\ell(e^{(n)})$	$\ell : E^{(n)} \rightarrow (\Sigma^*)^n$	its label, which is an n-tuple of strings
$w(e^{(n)})$	$w : E \rightarrow \mathcal{K}$	its weight, with $w(e^{(n)}) \neq \bar{0} \Leftrightarrow e^{(n)} \in E^{(n)}$, and
$n(e^{(n)})$	$n : E \rightarrow Q$	its target state

A *path* $\pi^{(n)}$ of length $r = |\pi^{(n)}|$ is a sequence of transitions $e_1^{(n)} e_2^{(n)} \cdots e_r^{(n)}$ such that $n(e_i^{(n)}) = p(e_{i+1}^{(n)})$ for all $i \in [\![1, r-1]\!]$. A path is said to be *successful* iff $p(e_1^{(n)}) \in I$ and $n(e_r^{(n)}) \in F$. In the following we consider only successful paths. The label of a successful path $\pi^{(n)}$ equals the concatenation of the labels of its transitions

$$\ell(\pi^{(n)}) \;=\; \ell(e_1^{(n)})\,\ell(e_2^{(n)})\;\cdots\;\ell(e_r^{(n)}) \tag{2}$$

and is an n-tuple of strings

$$\ell(\pi^{(n)}) \;=\; s^{(n)} \;=\; \langle s_1, s_2, \ldots, s_n \rangle \tag{3}$$

If all strings $s_j \in \Sigma^*$ (with $j \in [\![1, n]\!]$) of a tuple $s^{(n)}$ are equal, we use the short-hand notation $s_j^{(n)}$ on the terminal string s_j. For example:

$$(abc)^{(3)} \;=\; \langle abc, abc, abc \rangle \tag{4}$$

$$\varepsilon^{(4)} \;=\; \langle \varepsilon, \varepsilon, \varepsilon, \varepsilon \rangle \tag{5}$$

The n strings on any transition $e^{(n)}$ are not “bound” to each other. For example, the string triple $s^{(3)} = \langle aaa, bb, cccc \rangle$ can be encoded, among others, by any of the following sequences of transitions: $(a{:}b{:}cc)(a{:}b{:}c)(a{:}\varepsilon{:}c)$ or $(aa{:}\varepsilon{:}\varepsilon)(a{:}b{:}cc)(\varepsilon{:}b{:}cc)$ or $(aaa{:}bb{:}cccc)(\varepsilon{:}\varepsilon{:}\varepsilon)$, etc.

The weight $w(\pi^{(n)})$ of a successful path is

$$w(\,\pi^{(n)}\,) \;=\; \lambda(\,p(e_1^{(n)})\,) \;\otimes\; \left(\bigotimes_{j=[\![1,r]\!]} w(\,e_j^{(n)}\,) \right) \;\otimes\; \varrho(\,n(e_r^{(n)})\,) \tag{6}$$

We denote by $\Pi(A^{(n)})$ the (possibly infinite) set of successful paths of $A^{(n)}$ and by $\Pi(s^{(n)})$ the (possibly infinite) set of successful paths for the n-tuple of strings $s^{(n)}$

$$\Pi(s^{(n)}) \;=\; \{\; \pi^{(n)} \in \Pi(A^{(n)}) \mid s^{(n)} = \ell(\pi^{(n)}) \;\} \tag{7}$$

We call $\mathcal{R}(A^{(n)})$ the n-ary or n-tape relation of $A^{(n)}$. It is the (possibly infinite) set of n-tuples of strings $s^{(n)}$ having successful paths in $A^{(n)}$:

$$\mathcal{R}^{(n)} \;=\; \mathcal{R}(A^{(n)}) \;=\; \{\; s^{(n)} \mid \exists \pi^{(n)} \in \Pi(A^{(n)}) \wedge \ell(\pi^{(n)}) = s^{(n)} \;\} \tag{8}$$

The weight for any n-tuple of strings $s^{(n)} \in \mathcal{R}(A^{(n)})$ is the collection (semiring sum) of the weights of all paths labeled with $s^{(n)}$:

$$w(s^{(n)}) \;=\; \bigoplus_{\pi^{(n)} \in \Pi\left(s^{(n)}\right)} w(\pi^{(n)}) \tag{9}$$

By relation we mean simply a co-occurrence of strings in tuples. We do not assume any particular relation between those strings such as an input-output relation. All following operations and algorithms are independent from any particular relation. It is, however, possible to define an arbitrary weighted relation between the different tapes of $\mathcal{R}(A^{(n)})$. For example, $\mathcal{R}(A^{(2)})$ of a weighted *transducer* $A^{(2)}$ is usually considered as a weighted input-output relation between its two tapes, that are called *input tape* and *output tape*.

In the following we will not distinguish between a language $\mathcal{L}$ and a 1-tape relation $\mathcal{R}^{(1)}$, which allows us to define operations only on relations rather than on both languages and relations.

4 Operations

This section defines operations on string n-tuples and n-tape relations, taking their weights into account. Whenever these operations are used on transitions, paths, or automata, they are actually applied to their labels or relations respectively. For example, the binary operation $\ddot{\circ}$ on two automata, $A_1^{(n)} \ddot{\circ} A_2^{(n)}$, actually means $\mathcal{R}(A_1^{(n)} \ddot{\circ} A_2^{(n)}) = \mathcal{R}(A_1^{(n)}) \ddot{\circ} \mathcal{R}(A_2^{(n)})$. The unary operation $\dot{\circ}$ on one automaton, $\dot{\circ} A^{(n)}$, actually means $\mathcal{R}(\dot{\circ} A^{(n)}) = \dot{\circ} \mathcal{R}(A^{(n)})$.

Ultimately, we are interested in multi-tape intersection and transduction. The other operations are introduced because they serve as basis for the two.

4.1 Pairing and Concatenation

We define the *pairing* of two string tuples, $s^{(n)} : v^{(m)} = u^{(n+m)}$, and its weight as

$$\langle s_1, \ldots, s_n \rangle : \langle v_1, \ldots, v_m \rangle \quad =_{\text{def}} \quad \langle s_1, \ldots, s_n, v_1, \ldots, v_m \rangle \tag{10}$$

$$w\left(\langle s_1, \ldots, s_n \rangle : \langle v_1, \ldots, v_m \rangle \right) \quad =_{\text{def}} \quad w\left(\langle s_1, \ldots, s_n \rangle \right) \otimes w\left(\langle v_1, \ldots, v_m \rangle \right) \tag{11}$$

Pairing is associative (concerning both the string tuples and their weights) :

$$s_1^{(n_1)}{:}s_2^{(n_2)}{:}s_3^{(n_3)} \;=\; \left(s_1^{(n_1)}{:}s_2^{(n_2)}\right){:}s_3^{(n_3)} \;=\; s_1^{(n_1)}{:}\left(s_2^{(n_2)}{:}s_3^{(n_3)}\right) \;=\; s^{(n_1+n_2+n_3)} \tag{12}$$

We will not distinguish between 1-tuples of strings and strings, and hence, instead of $s^{(1)}{:}v^{(1)}$ or $\langle s\rangle{:}\langle v\rangle$, simply write $s{:}v$.

The *concatenation* of two string tuples of equal arity, $s^{(n)} v^{(n)} = u^{(n)}$, and its weight are defined as

$$\langle s_1, \ldots, s_n \rangle \langle v_1, \ldots, v_n \rangle \quad =_{\text{def}} \quad \langle s_1 v_1, \ldots, s_n v_n \rangle \tag{13}$$

$$w\left(\langle s_1, \ldots, s_n \rangle \langle v_1, \ldots, v_n \rangle \right) \quad =_{\text{def}} \quad w\left(\langle s_1, \ldots, s_n \rangle \right) \otimes w\left(\langle v_1, \ldots, v_n \rangle \right) \tag{14}$$

Concatenation is associative (concerning both the string tuples and their weights) :

$$s_1^{(n)} s_2^{(n)} s_3^{(n)} \;=\; \left(s_1^{(n)} s_2^{(n)}\right) s_3^{(n)} \;=\; s_1^{(n)} \left(s_2^{(n)} s_3^{(n)}\right) \;=\; s^{(n)} \tag{15}$$

Again, we will not distinguish between 1-tuples of strings and strings, and hence, instead of $s^{(1)} v^{(1)}$ or $\langle s\rangle\langle v\rangle$, simply write sv.

The relation retween pairing and concatenation can be expressed through a matrix of string tuples

$$\begin{bmatrix} s_{11}^{(n_1)} & \cdots & s_{1r}^{(n_1)} \\ \vdots & & \vdots \\ s_{m1}^{(n_m)} & \cdots & s_{mr}^{(n_m)} \end{bmatrix} \tag{16}$$

where the $s_{jk}^{(n_j)}$ are horizontally concatenated and vertically paired:

$$\begin{aligned} s^{(n_1+\ldots+n_m)} &= \left(s_{11}^{(n_1)} \cdots s_{1r}^{(n_1)}\right) \;:\; \cdots \;:\; \left(s_{m1}^{(n_m)} \cdots s_{mr}^{(n_m)}\right) \\ &= \left(s_{11}^{(n_1)} \;:\; \cdots \;:\; s_{m1}^{(n_m)}\right) \cdots \left(s_{1r}^{(n_1)} \;:\; \cdots \;:\; s_{mr}^{(n_m)}\right) \end{aligned} \tag{17}$$

Note, this equation does not hold for the weights of the $s_{jk}^{(n_j)}$, unless they are defined over a commutative semiring $\mathcal{K}$.

4.2 Cross-Product

The *cross-product* of two n-tape relations, $\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)} = \mathcal{R}^{(n+m)}$, is based on pairing and is defined as

$$\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)} \;=_{\text{def}}\; \{\, s^{(n)} : v^{(m)} \;\mid\; s^{(n)} \in \mathcal{R}_1^{(n)}, v^{(m)} \in \mathcal{R}_2^{(m)} \,\} \tag{18}$$

The weight of each string tuple $u^{(n+m)} \in \mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)}$ follows from the definition of pairing.

The cross product is an associative operation.

A well-know special case is the cross-product of two acceptors (1-tape automata) leading to a transducer (2-tape automaton) :

$$A^{(2)} \;=\; A_1^{(1)} \times A_2^{(1)} \tag{19}$$

$$\mathcal{R}(\, A^{(2)}\,) \;=\; \{\, s : v \;\mid\; s \in \mathcal{R}(A_1^{(1)}), v \in \mathcal{R}(A_2^{(1)})\,\} \tag{20}$$

$$w_A(\, s : v\,) \;=\; w_{A_1}(s) \;\otimes\; w_{A_2}(v) \tag{21}$$

4.3 Projection and Complementary Projection

The *projection*, $\mathcal{P}_{j,k,\dots}(s^{(n)})$, of a string tuple is defined as

$$\mathcal{P}_{j,k,\dots}(\, \langle s_1, \dots, s_n \rangle\,) \;=_{\text{def}}\; \langle s_j, s_k, \dots \rangle \tag{22}$$

It retains only those strings (i.e., tapes) of the tuple that are specified by the indices $j, k, \ldots \in [\![1, n]\!]$, and places them in the specified order. Projection indices can occur in any order and more that once. Thus the tapes of $s^{(n)}$ can, e.g., be reversed or duplicated:

$$\mathcal{P}_{n,\dots,1}(\, \langle s_1, \dots, s_n \rangle\,) \;=\; \langle s_n, \dots, s_1 \rangle \tag{23}$$

$$\mathcal{P}_{j,j,j}(\, \langle s_1, \dots, s_n \rangle\,) \;=\; \langle s_j, s_j, s_j \rangle \tag{24}$$

The weight of the n-tuple $s^{(n)}$ is not modified by the projection (if we consider $s^{(n)}$ not as a member of a relation).

The projection of an n-tape relation is the projection of all its string tuples:

$$\mathcal{P}_{j,k,\dots}(\mathcal{R}^{(n)}) \;=_{\text{def}}\; \{ v^{(m)} \;\mid\; \exists s^{(n)} \in \mathcal{R}^{(n)} \wedge \mathcal{P}_{j,k,\dots}(s^{(n)}) = v^{(m)} \} \tag{25}$$

The weight of each $v^{(m)} \in \mathcal{P}_{j,k,\dots}(\mathcal{R}^{(n)})$ is the collection (semiring sum) of the weights of each $s^{(n)} \in \mathcal{R}^{(n)}$ leading, when projected, to $v^{(m)}$:

$$w(v^{(m)}) \;=_{\text{def}} \bigoplus_{s^{(n)} \,\mid\, \mathcal{P}_{j,k,\dots}(s^{(n)}) = v^{(m)}} w(s^{(n)}) \tag{26}$$

The *complementary projection*, $\overline{\mathcal{P}}_{j,k,\dots}(s^{(n)})$, of a string n-tuple $s^{(n)}$ removes all those strings (i.e., tapes) of the tuple that are specified by the indices $j, k, \ldots \in [\![1, n]\!]$, and preserves all other strings in their original order.[1] It is defined as

$$\overline{\mathcal{P}}_{j,k,\dots}(\, \langle s_1, \dots, s_n \rangle\,) \;=_{\text{def}}\; \langle \dots, s_{j-1}, s_{j+1}, \dots, s_{k-1}, s_{k+1}, \dots \rangle \tag{27}$$

[1] Contrary to other authors, we do not call $\overline{\mathcal{P}}(\,)$ an *inverse projection* because it is not the inverse of a projection in the sense: $\alpha = \mathcal{P}(\beta)$ and $\beta = \mathcal{P}^{-1}(\alpha)$.

Complementary projection indices can occur in any order, but only once.

The complementary projection of an n-tape relation equals the complementary projection of all its string tuples:

$$\overline{\mathcal{P}}_{j,k,\ldots}(\mathcal{R}^{(n)}) \;=_{\text{def}}\; \{v^{(m)} \mid \exists s^{(n)} \in \mathcal{R}^{(n)} \wedge \overline{\mathcal{P}}_{j,k,\ldots}(s^{(n)}) = v^{(m)}\} \tag{28}$$

The weight of each $v^{(m)} \in \overline{\mathcal{P}}_{j,k,\ldots}(\mathcal{R}^{(n)})$ is the collection of the weights of each $s^{(n)} \in \mathcal{R}^{(n)}$ leading, when complementary projected, to $v^{(m)}$:

$$w(v^{(m)}) \;=_{\text{def}} \bigoplus_{s^{(n)} \mid \overline{\mathcal{P}}_{j,k,\ldots}(s^{(n)}) = v^{(m)}} w(s^{(n)}) \tag{29}$$

4.4 Auto-Intersection

We define the *auto-intersection* of a relation, $\mathcal{I}_{j,k}(\mathcal{R}^{(n)})$, on the tapes j and k as the subset of $\mathcal{R}^{(n)}$ that contains all $s^{(n)}$ with equal s_j and s_k:

$$\mathcal{I}_{j,k}(\;\mathcal{R}^{(n)}\;) \;=_{\text{def}}\; \{\; s^{(n)} \in \mathcal{R}^{(n)} \;\mid\; s_j = s_k \;\} \tag{30}$$

The weight of any $s^{(n)} \in \mathcal{I}_{j,k}(\mathcal{R}^{(n)})$ is not modified.

For example (Figure 1)

$$\mathcal{R}_1^{(3)} \;=\; \langle a,x,\varepsilon\rangle\,\langle b,y,a\rangle^*\,\langle \varepsilon,z,b\rangle \;=\; \{\,\langle ab^k, xy^kz, a^kb\rangle \;\mid\; k \in \mathbb{N}\,\} \tag{31}$$

$$\mathcal{I}_{1,3}(\mathcal{R}_1^{(3)}) \;=\; \{\,\langle ab^1, xy^1z, a^1b\rangle\,\} \tag{32}$$

Auto-intersection of regular n-tape relations is not necessarily regular. For example (Figure 3)

$$\mathcal{R}_2^{(3)} \;=\; \langle a,\varepsilon,x\rangle^*\,\langle a,a,y\rangle\,\langle \varepsilon,a,z\rangle^* \;=\; \{\,\langle a^ka, aa^h, x^kyz^h\rangle \;\mid\; k,h \in \mathbb{N}\,\} \tag{33}$$

$$\mathcal{I}_{1,2}(\mathcal{R}_2^{(3)}) \;=\; \{\,\langle a^ka, aa^k, x^kyz^k\rangle \;\mid\; k \in \mathbb{N}\,\} \tag{34}$$

The result is not regular because x^kyz^k is not regular.

4.5 Multi-Tape and Single-Tape Intersection

The multi-tape intersection of two multi-tape relations, $\mathcal{R}_1^{(n)}$ and $\mathcal{R}_2^{(m)}$, uses r tapes in each relation, and intersects them pair-wise. The operation pairs each string tuple $s^{(n)} \in \mathcal{R}_1^{(n)}$ with each string tuple $v^{(m)} \in \mathcal{R}_2^{(m)}$ iff $s_{j_i} = v_{k_i}$ with $j_i \in [\![1,n]\!], k_i \in [\![1,m]\!]$ for all $i \in [\![1,r]\!]$. Multi-tape intersection is defined as:

$$\mathcal{R}_1^{(n)} \bigcap_{\substack{j_1,\,k_1\\ \cdots\\ j_r,\,k_r}} \mathcal{R}_2^{(m)} \;=\; \mathcal{R}^{(n+m-r)} \tag{35}$$

$$=_{\text{def}}\; \{u^{(n+m-r)} \;\mid\; \exists s^{(n)} \in \mathcal{R}_1^{(n)}, \exists v^{(m)} \in \mathcal{R}_2^{(m)},\; s_{j_i} = v_{k_i},\; j_i \in [\![1,n]\!], k_i \in [\![1,m]\!], \forall i \in [\![1,r]\!]$$
$$u^{(n+m-r)} = \overline{\mathcal{P}}_{n+k_1,\ldots,n+k_r}(s^{(n)}{:}v^{(m)})\}$$

All tapes k_i of $\mathcal{R}_2^{(m)}$ that have directly participated in the intersection are afterwards equal to the tapes j_i of $\mathcal{R}_1^{(n)}$, and are removed. All tapes j_i are kept for possible reuse by subsequent operations. All other tapes of both relations are preserved without modification.

The weight of each $u^{(n+m-r)} \in \mathcal{R}^{(n+m-r)}$ is

$$w(\ u^{(n+m-r)}\ ) \ = \ w(s^{(n)}) \ \otimes \ w(v^{(m)}) \tag{36}$$

This weight follows only from pairing (Eq. 11). It is not influenced by complementary projection (Eq. 29) because any two $u^{(n+m)} = s^{(n)}{:}v^{(m)}$ that differ in v_{k_i} also differ in s_{j_i}, and hence cannot become equal when the v_{k_i} are removed.

The multi-tape intersection of two relations, $\mathcal{R}_1^{(n)}$ and $\mathcal{R}_2^{(m)}$, can be compiled by

$$\mathcal{R}_1^{(n)} \underset{\substack{j_1,\,k_1\\ \cdots\\ j_r,\,k_r}}{\cap} \mathcal{R}_2^{(m)} \ = \ \overline{\mathcal{P}}_{n+k_1,\ldots,n+k_r}\left(\ \mathcal{I}_{j_r,n+k_r}(\ \cdots\ \mathcal{I}_{j_1,n+k_1}(\ \mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)}\ )\ \cdots\ )\ \right) \tag{37}$$

as can been seen from

$$\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)} \ = \ \{\ s^{(n)}{:}v^{(m)} \ \mid \ s^{(n)} \in \mathcal{R}_1^{(n)}, v^{(m)} \in \mathcal{R}_2^{(m)}\ \} \tag{38}$$

$$\mathcal{I}_{j_1,n+k_1}(\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)}) \ = \ \{\ s^{(n)}{:}v^{(m)} \ \mid \ \exists s^{(n)} \in \mathcal{R}_1^{(n)}, \exists v^{(m)} \in \mathcal{R}_2^{(m)}, s_{j_1} = v_{k_1}\ \} \tag{39}$$

$$etc.$$

Multi-tape intersection is a generalization of classical intersection of transducers which is known to be not necessarily regular (Rabin and Scott, 1959) :

$$A_1^{(2)} \cap A_2^{(2)} \ = \ A_1^{(2)} \underset{\substack{1,\,1\\ 2,\,2}}{\cap} A_2^{(2)} \ = \ \overline{\mathcal{P}}_{3,4}\left(\ \mathcal{I}_{2,4}(\ \mathcal{I}_{1,3}(\ A_1^{(2)} \times A_2^{(2)}\ )\ )\ \right) \tag{40}$$

Consequently, multi-tape intersection has the same property. In our approach this results from the potential non-regularity of auto-intersection (Eq. 37).

We speak about *single-tape intersection* if only one tape is used in each relation ($r = 1$). A well-known special case is the intersection of two acceptors (1-tape automata) leading to an acceptor

$$A_1^{(1)} \cap A_2^{(1)} \ = \ A_1^{(1)} \underset{1,1}{\cap} A_2^{(1)} \ = \ \overline{\mathcal{P}}_2\left(\ \mathcal{I}_{1,2}(\ A_1^{(1)} \times A_2^{(1)}\ )\ \right) \tag{41}$$

and yielding the relation

$$\mathcal{R}\left(\ A_1^{(1)} \cap A_2^{(1)}\ \right) \ = \ \{\ s \ \mid \ s \in \mathcal{R}(A_1)\,,\ s \in \mathcal{R}(A_2)\ \} \tag{42}$$

$$w(s) \ = \ w_{A_1}(s) \otimes w_{A_2}(s) \tag{43}$$

Another well-known special case is the composition of two transducers (2-tape automata) leading to a transducer. Here, we need, however, an additional complementary projection:[2]

$$A_1^{(2)} \diamond A_2^{(2)} \ = \ \overline{\mathcal{P}}_2(\ A_1^{(2)} \underset{2,1}{\cap} A_2^{(2)}\ ) \ = \ \overline{\mathcal{P}}_{2,3}\left(\ \mathcal{I}_{2,3}(\ A_1^{(2)} \times A_2^{(2)}\ )\ \right) \tag{44}$$

It yields the relation:

$$\mathcal{R}\left(A_1^{(2)} \diamond A_2^{(2)}\right) \ = \ \{u^{(2)} \mid \exists s^{(2)} {\in} \mathcal{R}(A_1^{(2)}), \exists v^{(2)} {\in} \mathcal{R}(A_2^{(2)}), s_2 {=} v_1, u^{(2)} {=} \overline{\mathcal{P}}_{2,3}(s^{(2)}{:}v^{(2)})\} \tag{45}$$

$$w(u^{(2)}) \ = \ \bigoplus_{s^{(2)},v^{(2)} \mid u_1=s_1, s_2=v_1, v_2=u_2} w_{A_1}(s^{(2)}) \ \otimes \ w_{A_2}(v^{(2)}) \tag{46}$$

Multi-tape and single-tape intersection are neither associative nor commutative, except for special cases with $r = n = m$, such as the above intersection of acceptors and transducers.

[2] Composition of transducers T_i is expressed either by the $\diamond$ or the $\circ$ operator. However, $T_1 \diamond T_2$ equals $T_2 \circ T_1$ which corresponds to $\mathrm{T}_2(\,\mathrm{T}_1(\ \ )\,)$ in functional notation (Birkhoff and Bartee, 1970).

4.6 Transduction

A WMTA, $A^{(n)}$, can be used as a transducer having r input tapes, j_1 to j_r, and x output tapes, k_1 to k_x, which do not have to be consecutive or disjoint.

To apply $A^{(n)}$ to a weighted r-tuple of input strings, the tuple $s^{(r)}$ is converted into an input WMTA, $I^{(r)}$, having one single path labeled with $s^{(r)}$ and weighted with $w(s^{(r)})$. An output WMTA, $O^{(x)}$, whose relation contains all weighted x-tuples of output strings, $v^{(x)}$, is then obtained through multitape-intersection and projection:

$$O^{(x)} \;=\; \mathcal{P}_{k_1,\ldots,k_x}\big(\, A^{(n)} \bigcap_{\substack{j_1,1 \\ \cdots \\ j_r,r}} I^{(r)} \,\big) \tag{47}$$

5 Example of Classical Transducer Intersection

The following example of classical transducer intersection of $A_1^{(2)}$ and $A_2^{(2)}$ is regular:[3]

$$\begin{array}{cc} \mathrm{a} & \mathrm{b} \\ \varepsilon & \mathrm{A} \end{array} \begin{pmatrix} \mathrm{c} & \mathrm{a} & \mathrm{b} \\ \mathrm{B} & \varepsilon & \mathrm{C} \end{pmatrix}^{*} \begin{array}{ccccc} \varepsilon & \varepsilon & \varepsilon & \mathrm{c} & \varepsilon \\ \mathrm{A} & \mathrm{B} & \mathrm{C} & \varepsilon & \mathrm{A} \end{array} \quad \bigcap_{\substack{1,1 \\ 2,2}} \quad \begin{array}{c} \varepsilon \\ \mathrm{A} \end{array} \begin{pmatrix} \mathrm{a} & \mathrm{b} & \varepsilon & \mathrm{c} \\ \mathrm{B} & \varepsilon & \mathrm{C} & \mathrm{A} \end{pmatrix}^{*}$$

It has one theoretical solution which is

$$\begin{array}{cc} \mathrm{a} & \mathrm{b} \\ \varepsilon & \mathrm{A} \end{array} \begin{pmatrix} \mathrm{c} & \mathrm{a} & \mathrm{b} \\ \mathrm{B} & \varepsilon & \mathrm{C} \end{pmatrix}^{1} \begin{array}{ccccc} \varepsilon & \varepsilon & \varepsilon & \mathrm{c} & \varepsilon \\ \mathrm{A} & \mathrm{B} & \mathrm{C} & \varepsilon & \mathrm{A} \end{array} = \begin{array}{ccccccc} \mathrm{a} & \mathrm{b} & \mathrm{c} & \mathrm{a} & \mathrm{b} & \mathrm{c} & \varepsilon \\ \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{A} \end{array} = \begin{array}{c} \varepsilon \\ \mathrm{A} \end{array} \begin{pmatrix} \mathrm{a} & \mathrm{b} & \varepsilon & \mathrm{c} \\ \mathrm{B} & \varepsilon & \mathrm{C} & \mathrm{A} \end{pmatrix}^{2}$$

This solution cannot be compiled with any of the above mentioned previous approaches (Section 2). It cannot be enabled by any pre-transformation of the WMTAs that does not change their relations, $\mathcal{R}(A_1^{(2)})$ and $\mathcal{R}(A_2^{(2)})$. All above mentioned approaches do not exceed the following alternatives.

5.1 First Failing Alternative

One can start by typing all symbols (and ε) with respect to the tapes, to make the alphabets of different tapes disjoint (which can be omitted for symbols occurring on one tape only) :

$$\begin{array}{cc} \mathrm{a} & \mathrm{b} \\ \varepsilon_2 & \mathrm{A} \end{array} \begin{pmatrix} \mathrm{c} & \mathrm{a} & \mathrm{b} \\ \mathrm{B} & \varepsilon_2 & \mathrm{C} \end{pmatrix}^{*} \begin{array}{ccccc} \varepsilon_1 & \varepsilon_1 & \varepsilon_1 & \mathrm{c} & \varepsilon_1 \\ \mathrm{A} & \mathrm{B} & \mathrm{C} & \varepsilon_2 & \mathrm{A} \end{array} \quad \bigcap_{\substack{1,1 \\ 2,2}} \quad \begin{array}{c} \varepsilon_1 \\ \mathrm{A} \end{array} \begin{pmatrix} \mathrm{a} & \mathrm{b} & \varepsilon_1 & \mathrm{c} \\ \mathrm{B} & \varepsilon_2 & \mathrm{C} & \mathrm{A} \end{pmatrix}^{*}$$

Then, one converts n tapes into 1 tape, such that each transition, labeled with n symbols, is transformed into a sequence of n transitions, labeled with 1 symbol each, which is equivalent to Ganchev's approach (Ganchev, Mihov, and Schulz, 2003) :

$$\mathrm{a}\ \varepsilon_2\mathrm{b}\ \mathrm{A}\ \big(\mathrm{c}\ \mathrm{B}\,\mathrm{a}\ \varepsilon_2\mathrm{b}\ \mathrm{C}\big)^{*}\ \varepsilon_1\mathrm{A}\,\varepsilon_1\mathrm{B}\,\varepsilon_1\mathrm{C}\,\mathrm{c}\ \varepsilon_2\varepsilon_1\mathrm{A} \quad \cap \quad \varepsilon_1\mathrm{A}\ \big(\mathrm{a}\ \mathrm{B}\,\mathrm{b}\ \varepsilon_2\varepsilon_1\mathrm{C}\,\mathrm{c}\ \mathrm{A}\big)^{*}$$

After these transformations, it is not possible to obtain the above theoretical solution by means of classical intersection of 1-tape automata, even not after ε-removal:

$$\mathrm{a}\ \mathrm{b}\ \mathrm{A}\ \big(\mathrm{c}\ \mathrm{B}\ \mathrm{a}\ \mathrm{b}\ \mathrm{C}\big)^{*}\ \mathrm{A}\ \mathrm{B}\ \mathrm{C}\ \mathrm{c}\ \mathrm{A} \quad \cap \quad \mathrm{A}\ \big(\mathrm{a}\ \mathrm{B}\ \mathrm{b}\ \mathrm{C}\ \mathrm{c}\ \mathrm{A}\big)^{*}$$

[3]For sake of space and clarity we represent all regular expressions in this section in a special form where each tape appears on a different row and symbols of the same transition are vertically aligned. Note that it is not a matrix representation. More conventionally $A_1^{(2)}$ could be written as $\langle a,\varepsilon\rangle\langle b,A\rangle\ (\ \langle c,B\rangle\langle a,\varepsilon\rangle\langle b,C\rangle\ )^{*}\ \langle\varepsilon,A\rangle\langle\varepsilon,B\rangle\langle\varepsilon,C\rangle\langle c,\varepsilon\rangle\langle\varepsilon,A\rangle$.

5.2 Second Failing Alternative

Alternatively, one could start with synchronizing the WMTAs. This is not possible across a whole WMTA, but only within "limited sections": in our example this means before, inside, and after the cycles:

$$\begin{array}{cc} \mathrm{a} & \mathrm{b} \\ \mathrm{A} & \varepsilon \end{array} \left(\begin{array}{ccc} \mathrm{c} & \mathrm{a} & \mathrm{b} \\ \mathrm{B} & \mathrm{C} & \varepsilon \end{array} \right)^{*} \begin{array}{cccc} \mathrm{c} & \varepsilon & \varepsilon & \varepsilon \\ \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{A} \end{array} \quad \underset{\substack{1,1 \\ 2,2}}{\cap} \quad \begin{array}{c} \varepsilon \\ \mathrm{A} \end{array} \left(\begin{array}{ccc} \mathrm{a} & \mathrm{b} & \mathrm{c} \\ \mathrm{B} & \mathrm{C} & \mathrm{A} \end{array} \right)^{*}$$

Then, one can proceed as before by first typing the symbols with respect to the tapes

$$\begin{array}{cc} \mathrm{a} & \mathrm{b} \\ \mathrm{A} & \varepsilon_2 \end{array} \left(\begin{array}{ccc} \mathrm{c} & \mathrm{a} & \mathrm{b} \\ \mathrm{B} & \mathrm{C} & \varepsilon_2 \end{array} \right)^{*} \begin{array}{cccc} \mathrm{c} & \varepsilon_1 & \varepsilon_1 & \varepsilon_1 \\ \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{A} \end{array} \quad \underset{\substack{1,1 \\ 2,2}}{\cap} \quad \begin{array}{c} \varepsilon_1 \\ \mathrm{A} \end{array} \left(\begin{array}{ccc} \mathrm{a} & \mathrm{b} & \mathrm{c} \\ \mathrm{B} & \mathrm{C} & \mathrm{A} \end{array} \right)^{*}$$

and then transforming n tapes into 1 tape

$$\mathrm{a\ A\ b}\ \varepsilon_2 \left(\mathrm{c\ B\ a\ C\ b}\ \varepsilon_2 \right)^{*} \mathrm{c\ A}\, \varepsilon_1 \mathrm{B}\, \varepsilon_1 \mathrm{C}\, \varepsilon_1 \mathrm{A} \quad \cap \quad \varepsilon_1 \mathrm{A} \left(\mathrm{a\ B\ b\ C\ c\ A} \right)^{*}$$

The solution cannot be compiled with this alternative either, even not after ε-removal:

$$\mathrm{a\ A\ b} \left(\mathrm{c\ B\ a\ C\ b} \right)^{*} \mathrm{c\ A\ B\ C\ A} \quad \cap \quad \mathrm{A} \left(\mathrm{a\ B\ b\ C\ c\ A} \right)^{*}$$

5.3 Solution with Our Approach

To compile multi-tape intersection according to the above procedure (Eq. 37)

$$A^{(2)} \;=\; A_1^{(2)} \underset{\substack{1,1 \\ 2,2}}{\cap} A_2^{(2)} \;=\; \overline{\mathcal{P}}_{3,4}(\; \mathcal{I}_{2,4}(\; \mathcal{I}_{1,3}(\; A_1^{(2)} \times A_2^{(2)} \;)\;)\;) \tag{48}$$

we proceed in 3 steps. First, we compile $B_1^{(4)} = \mathcal{I}_{1,3}(A_1^{(2)} \times A_2^{(2)})$ in one single step with an algorithm that follows the principle of transducer composition and simulates the behaviour of Mohri's ε-filter (Section 6.3).[4] For the above example, we obtain

$$\begin{array}{ccc} \varepsilon & \mathrm{a} & \mathrm{b} \\ \varepsilon & \varepsilon & \mathrm{A} \\ \varepsilon & \mathrm{a} & \mathrm{b} \\ \mathrm{A} & \mathrm{B} & \varepsilon \end{array} \left(\begin{array}{cccc} \varepsilon & \mathrm{c} & \mathrm{a} & \mathrm{b} \\ \varepsilon & \mathrm{B} & \varepsilon & \mathrm{C} \\ \varepsilon & \mathrm{c} & \mathrm{a} & \mathrm{b} \\ \mathrm{C} & \mathrm{A} & \mathrm{B} & \varepsilon \end{array} \right)^{*} \begin{array}{ccccc} \varepsilon & \varepsilon & \varepsilon & \mathrm{c} & \varepsilon \\ \mathrm{A} & \mathrm{B} & \mathrm{C} & \varepsilon & \mathrm{A} \\ \varepsilon & \varepsilon & \varepsilon & \mathrm{c} & \varepsilon \\ \mathrm{C} & \varepsilon & \varepsilon & \mathrm{A} & \varepsilon \end{array}$$

Next, we compile $B_2^{(4)} = \mathcal{I}_{2,4}(B_1^{(4)})$ using our auto-intersection algorithm (Section 6.2)

$$\begin{array}{ccc} \varepsilon & \mathrm{a} & \mathrm{b} \\ \varepsilon & \varepsilon & \mathrm{A} \\ \varepsilon & \mathrm{a} & \mathrm{b} \\ \mathrm{A} & \mathrm{B} & \varepsilon \end{array} \left(\begin{array}{cccc} \varepsilon & \mathrm{c} & \mathrm{a} & \mathrm{b} \\ \varepsilon & \mathrm{B} & \varepsilon & \mathrm{C} \\ \varepsilon & \mathrm{c} & \mathrm{a} & \mathrm{b} \\ \mathrm{C} & \mathrm{A} & \mathrm{B} & \varepsilon \end{array} \right)^{1} \begin{array}{ccccc} \varepsilon & \varepsilon & \varepsilon & \mathrm{c} & \varepsilon \\ \mathrm{A} & \mathrm{B} & \mathrm{C} & \varepsilon & \mathrm{A} \\ \varepsilon & \varepsilon & \varepsilon & \mathrm{c} & \varepsilon \\ \mathrm{C} & \varepsilon & \varepsilon & \mathrm{A} & \varepsilon \end{array}$$

[4] Composition with ε-filter has been shown to work on arbitrary transducers (Mohri, Pereira, and Riley, 1998).

and finally, $A^{(2)} = \overline{\mathcal{P}}_{3,4}(\ B_2^{(4)}\ )$ with a simple algorithm for complementary projection:

$$\begin{matrix} \varepsilon & \text{a} & \text{b} \\ \varepsilon & \varepsilon & \text{A} \end{matrix} \begin{pmatrix} \varepsilon & \text{c} & \text{a} & \text{b} \\ \varepsilon & \text{B} & \varepsilon & \text{C} \end{pmatrix}^1 \begin{matrix} \varepsilon & \varepsilon & \varepsilon & \text{c} & \varepsilon \\ \text{A} & \text{B} & \text{C} & \varepsilon & \text{A} \end{matrix}$$

This final result equals the theoretical solution.

6 Algorithms

In this section we propose and recall algorithms for the above defined operations on WMTAs: cross-product, auto-intersection, single-tape and multi-tape intersection. By convention, our WMTAs have only one initial state $i \in I$, without loss of generality, since for any WMTA with multiple initial states there exists a WMTA with a single initial state accepting the same relation.

We will use the following variables and definitions. The variables $\nu[q]$, $\mu[q]$, etc. serve for assigning temporarily additional data to a state q.

A_j	$= \langle \Sigma_j, Q_j, i_j, F_j, E_j, \mathcal{K}_j \rangle$	Original weighted automaton from which we will construct a new weighted automaton A
A	$= \langle \Sigma, Q, i, F, E, \mathcal{K} \rangle$	New weighted automaton resulting from a construction
$\nu[q]$	$= q_1$	State q_1 of an original automaton A_1 assigned to a state q of a new automaton A
$\mu[q]$	$= (q_1, q_2)$	pair of states (q_1, q_2) of two original automata, A_1 and A_2, assigned to a state q of a new automaton A
$\vartheta[q]$	$= (q_1, q_2, q_\varepsilon)$	triple of states belonging to the two original automata, A_1 and A_2, and to a simulated filter automaton, A_ε, respectively; assigned to a state q of a new automaton A
$\xi[q]$	$= (s, u)$	Pair of “leftover” substrings (s, u) assigned to a state q of a new automaton A
$\delta(s, u)$	$= \lvert s \rvert - \lvert u \rvert$	Delay between two string (or leftover substrings) s and u. For example: $\delta(\xi[q])$ also written as $\delta(q)$
$\chi[q]$	$= (\chi_1, \chi_2)$	Pair of integers assigned to a state q, expressing the lengths of two strings s and u on different tapes of the same path ending at q
lcp(s, s')		Longest common prefix of the strings s and s'
$\ell_{j,k,\ldots}(e)$	$= \mathcal{P}_{j,k,\ldots}(\ \ell(e)\ )$	Short-hand notation for the projection of the label of e

6.1 Cross Product

We describe two alternative algorithms to compile the cross product of two WMTAs, $A_1^{(n)}$ and $A_2^{(m)}$. The second algorithm is almost identical to classical algorithms for crossproduct of automata. Nevertheless, we recall it to make this report more complete and self-standing.

6.1.1 Conditions

Both algorithms require the semirings of the two original automata, $A_1^{(n)}$ and $A_2^{(m)}$, to be equal ($\mathcal{K}_1 = \mathcal{K}_2$). The second algorithm requires the common semiring $\mathcal{K} = \mathcal{K}_1 = \mathcal{K}_2$ to be commutative.

6.1.2 Algorithms

Cross product through path concatenation: The first algorithm pairs the label of each transition $e_1 \in E_1$ with $\varepsilon^{(m)}$ (producing $\ell(e_1):\varepsilon^{(m)}$), and the label of each transition $e_2 \in E_2$ with $\varepsilon^{(n)}$ (producing $\varepsilon^{(n)}:\ell(e_2)$), and then concatenates $A_1^{(n+m)}$ with $A_2^{(n+m)}$. We will refer to it as CrossPC(A_1, A_2) where the suffix *PC* stands for *path concatenation*.

CrossPC**(**$A_1^{(n)}, A_2^{(m)}$**)** $\rightarrow A$ **:**
1 $A \leftarrow \langle \Sigma_1 \cup \Sigma_2, Q_1 \cup Q_2, i_1, F_2, E_1 \cup E_2, \mathcal{K}_1 \rangle$
2 for $\forall e_1 \in E_1$ do
3 $\quad \ell(e_1) \leftarrow \ell(e_1):\varepsilon^{(m)}$
4 for $\forall e_2 \in E_2$ do
5 $\quad \ell(e_2) \leftarrow \varepsilon^{(n)}:\ell(e_2)$
6 for $\forall q \in F_1$ do
7 $\quad E \leftarrow E \cup \{ \langle q, \varepsilon^{(n+m)}, \varrho(q), i_2 \rangle \}$
8 $\quad \varrho(q) \leftarrow \bar{0}$
9 return A

We start with a WMTA A that is equipped with the union of the alphabets, the union of the state sets, and the union of the transition sets of A_1 and A_2. The initial state of A equals that A_1, its set of final states equals that of A_2, and its semiring equals those of A_1 and A_2 (Line 1). First, we (post-) pair the labels of all transitions originally coming from A_1 with $\varepsilon^{(m)}$, and (pre-) pair the labels of all transition from A_2 with $\varepsilon^{(n)}$. Then, we connect all final states of A_1 with the initial state of A_2 through $\varepsilon^{(n+m)}$-transitions, as is usually done in the concatenation of automata.

The disadvantages of this algorithm are that the paths of A become longer than in the second algorithm below and that each transition of A is partially labeled with ε, which may increase the running time of subsequently applied operations.

To adapt this algorithm to non-weighted MTAs, one has to remove the weight from Line 7 and replace Line 8 with: $Final(q) \leftarrow$ *false*.

Cross product through path alignment: The second algorithm pairs each string tuple of $A_1^{(n)}$ with each string tuple of $A_2^{(m)}$, following the definition (Eq. 18). The algorithm actually pairs each path π_1 of $A_1^{(n)}$ with each path π_2 of $A_2^{(m)}$ transition-wise, and appends ε-transitions to the shorter of two paired paths, so that both have equal length. We will refer to this algorithm as CrossPA(A_1, A_2) where the suffix *PA* stands for *path alignment*.

We start with a WMTA A whose alphabet is the union of the alphabets of A_1 and A_2, whose semiring equals those of A_1 and A_2, and that is otherwise empty (Line 1). First, we create the initial state i of A from the initial states of A_1 and A_2, and push i onto the stack (Lines 3, 20–26). While the stack is not empty, we take states q from it and access the states q_1 and q_2 that are assigned to q through $\mu[q]$ (Lines 4, 5).

CROSSPA$(A_1^{(n)}, A_2^{(m)}) \rightarrow A$ **:**
1 $A \leftarrow \langle \Sigma_1 \cup \Sigma_2, \emptyset, \bot, \emptyset, \emptyset, \mathcal{K}_1 \rangle$
2 $Stack \leftarrow \emptyset$
3 $i \leftarrow$ GETSTATE(i_1, i_2)
4 while $Stack \neq \emptyset$ do
5 $q \leftarrow$ *pop*$(Stack)$: $\mu[q] = (q_1, q_2)$
6 if $q_1 \neq \bot \ \wedge \ q_2 \neq \bot$
7 then for $\forall e_1 \in E(q_1)$ do
8 for $\forall e_2 \in E(q_2)$ do
9 $q' \leftarrow$ GETSTATE$(n(e_1), n(e_2))$
10 $E \leftarrow E \cup \{ \langle q,\ \ell(e_1) : \ell(e_2),\ w(e_1) \otimes w(e_2),\ q' \rangle \}$
11 if $\varrho(q_1) \neq \bar{0} \ \vee \ q_1 = \bot$
12 then for $\forall e_2 \in E(q_2)$ do
13 $q' \leftarrow$ GETSTATE$(\bot, n(e_2))$
14 $E \leftarrow E \cup \{ \langle q,\ \varepsilon^{(n)} : \ell(e_2),\ \varrho(q_1) \otimes w(e_2),\ q' \rangle \}$
15 if $\varrho(q_2) \neq \bar{0} \ \vee \ q_2 = \bot$
16 then for $\forall e_1 \in E(q_1)$ do
17 $q' \leftarrow$ GETSTATE$(n(e_1), \bot)$
18 $E \leftarrow E \cup \{ \langle q,\ \ell(e_1) : \varepsilon^{(m)},\ w(e_1) \otimes \varrho(q_2),\ q' \rangle \}$
19 return A

GETSTATE$(q_1, q_2) \rightarrow q$ **:**
20 if $\exists q' \in Q : \mu[q'] = (q_1, q_2)$
21 then $q \leftarrow q'$
22 else $Q \leftarrow Q \cup \{q\}$ [create new state]
23 $\varrho(q) \leftarrow \varrho(q_1) \otimes \varrho(q_2)$
24 $\mu[q] \leftarrow (q_1, q_2)$
25 *push*$(Stack, q)$
26 return q

If both q_1 and q_2 are defined ($\neq \bot$), we pair each outgoing transition e_1 of q_1 with each outgoing transition e_2 of q_2 (Lines 6–8), and create a transition in A (Line 10) whose label is the pair $\ell(e_1) : \ell(e_2)$ and whose target q' corresponds to the tuple of targets $(n(e_1), n(e_2))$ (Line 9). If q' does not exist yet, it is created and pushed onto the stack (Lines 20–26).

If we encounter a final state q_1 (with $\varrho(q_1) \neq \bar{0}$) in A_1, we follow the path beyond q_1 on an ε-transition that exists only "virtually" but not "physically" in A_1 (Lines 11, 12). The target of the resulting transition in A corresponds to the tuple of targets $(n(e_1), n(e_2))$ with $n(e_1)$ being undefined $(= \bot)$ because e_1 does not exist physically (Line 13). If we encounter a final state q_2 (with $\varrho(q_2) \neq \bar{0}$) in A_2, we proceed similarly (Lines 15–18).

The final weight of an undefined state $q = \bot$ is assumed to be $\bar{1}$: $\varrho(\bot) = \bar{1}$.

To adapt this algorithm to non-weighted MTAs, one has to remove the weights from the Lines 10, 14, and 18, and replace Line 23 with: $Final(q) \leftarrow Final(q_1) \wedge Final(q_2)$.

6.2 Auto-Intersection

We propose an algorithm that attempts to constructs the auto-intersection $A^{(n)}$ of a WMTA $A_1^{(n)}$. Our approach has some minor similarity with synchronization algorithms for transducers (Frougny and Sakarovitch, 1993; Mohri, 2003) : it uses the concept of delay between two tapes and assigns leftover-strings to states (see above).

In the context of our approach, we understand by *construction* the compilation of reachable states q and transitions $e^{(n)}$ of $A^{(n)}$, such that the absolute value of the delay $\delta(q)$, regarding tape j and k, does not exceed a limit $\delta_{\max 2}$ at any state q, i.e.: $\forall q : |\delta(q)| \leq \delta_{\max 2} \wedge q\ \mathsf{reachable}$. The limit $\delta_{\max 2}$ is imposed, i.e., any state whose delay would exceed it is not constructed.

We distinguish two cases. In the first case, the delay of none of the reachable and coreachable states exceeds a limit $\delta_{\max}$ (with $\delta_{\max} \leq \delta_{\max 2}$), i.e.: $\nexists q : \delta_{\max} < |\delta(q)| \leq \delta_{\max 2} \wedge q\ \mathsf{reachable} \wedge q\ \mathsf{coreachable}$. We call it a construction with *bounded delay* or a *successful* construction because it is guarantied to generate the attempted result $A^{(n)} = \mathcal{I}_{j,k}(A_1^{(n)})$. In this case the relation $\mathcal{I}_{j,k}(A_1^{(n)})$ has bounded delay, too, and is rational.[5] The limit $\delta_{\max}$ is not imposed, i.e., any state q whose delay exceeds it would still be constructed (which places the construction into the second case if q becomes coreachable).

In the second case, the delay of reachable and coreachable states is potentially unbounded. It exceeds $\delta_{\max}$, and would actually exceed any limit if it was not (brute-force) delimited by $\delta_{\max 2}$, i.e.: $\exists q : \delta_{\max} < |\delta(q)| \leq \delta_{\max 2} \wedge q\ \mathsf{reachable} \wedge q\ \mathsf{coreachable}$. We call this a construction with *potentially unbounded delay*. It is not successful, and we cannot conclude on the correctness of the result $A^{(n)}$ and on the boundedness and rationality of the relation $\mathcal{I}_{j,k}(A_1^{(n)})$.

We will first describe the algorithm and then present some examples for further illustration.

6.2.1 Algorithm

Our algorithm starts with the compilation of the limits $\delta_{\max}$ and $\delta_{\max 2}$, then proceeds with the construction of $A^{(n)}$, and finally verifies the success of the construction, according to the above conditions.

Compilation of limits: First, we traverse $A_1^{(n)}$ recursively, without traversing any state more than once, and record three values: $\widehat{\delta}_{max}$, being the maximal delay at any state, $\widehat{\delta}_{min}$, the minimal delay at any state, and $\widehat{\delta}_{cyc}$, the maximal absolute value of the delay of any cycle (Lines 3, 8–17). To do so, we assign to each state q_1 of $A_1^{(n)}$ a variable $\chi[q_1] = (\chi_1, \chi_2)$ with the above defined meaning. The delay at a state q_1 is $\delta(q_1) = \chi_1 - \chi_2$ (Lines 8, 9). The delay of a cycle on q_1 is the difference between $\delta'(q_1)$ at the end and $\delta(q_1)$ at the beginning of the cycle (Line 11).

Then, we compile δ_{cyc}, the maximal absolute value of delay required to match any two cycles. For example, let $\mathcal{R}(A_1^{(2)}) = (\{\langle aa, \varepsilon\rangle\} \cup \{\langle \varepsilon, aaa\rangle\})^*$, encoded by two cycles. To obtain a match between $\ell_1(\pi)$ and $\ell_2(\pi)$ of a path π of $A^{(2)} \subseteq \mathcal{I}_{1,2}(A_1^{(2)})$, we have to traverse the first cycle 3 times and the second two times, allowing for any permutation: $A^{(2)} = (\langle aa, \varepsilon\rangle^3 \langle \varepsilon, aaa\rangle^2 \cup \langle aa, \varepsilon\rangle^2 \langle \varepsilon, aaa\rangle^2 \langle aa, \varepsilon\rangle^1 \cup \ldots)^*$. This illustrates that in a match between any two cycles of $A_1^{(n)}$, the absolute value of the delay does not exceed $\delta_{cyc} = \widehat{\delta}_{\text{cyc}} \cdot max(\, 1,\ \widehat{\delta}_{\text{cyc}} - 1\,)$ (Line 4).

[5] A *rational* relation is a weighted regular relation.

---

getMaxDelays$(A_1, j, k) \rightarrow (\delta_{max}, \delta_{max2})$:
1 for $\forall q_1 \in Q_1$ do
2 $\quad\chi[q_1] \leftarrow \bot$
3 $(\widehat{\delta}_{max}, \widehat{\delta}_{min}, \widehat{\delta}_{cyc}) \leftarrow$ maxDel$(i_1, j, k, (0,0), (0,0,0))$
4 $\delta_{cyc} \leftarrow \widehat{\delta}_{cyc} \cdot max(1\ ,\ \widehat{\delta}_{cyc} - 1)$
5 $\delta_{max} \leftarrow max(\delta_{cyc}\ ,\ \widehat{\delta}_{max} - \widehat{\delta}_{min})$
6 $\delta_{max2} \leftarrow \delta_{max} + \delta_{cyc}$
7 return $(\delta_{max}\ ,\ \delta_{max2})$

maxDel$(q_1, j, k, (\chi'_1, \chi'_2), (\delta'_{max}, \delta'_{min}, \delta'_{cyc})) \rightarrow (\widehat{\delta}_{max}, \widehat{\delta}_{min}, \widehat{\delta}_{cyc})$:
8 $\widehat{\delta}_{\max} \leftarrow max(\delta'_{max}\ ,\ \chi'_1 - \chi'_2)$
9 $\widehat{\delta}_{\min} \leftarrow min(\delta'_{min}\ ,\ \chi'_1 - \chi'_2)$
10 if $\chi[q_1] = (\chi_1, \chi_2) \neq \bot$ [cycle end reached]
11 $\quad$ then $\widehat{\delta}_{cyc} \leftarrow max(\delta'_{cyc}\ ,\ \mid (\chi'_1 - \chi'_2) - (\chi_1 - \chi_2) \mid)$
12 $\quad$ else $\chi[q_1] \leftarrow (\chi'_1, \chi'_2)$
13 $\quad\quad \widehat{\delta}_{cyc} \leftarrow \delta'_{cyc}$
14 $\quad\quad$ for $\forall e \in E(q_1)$ do
15 $\quad\quad\quad (\widehat{\delta}_{max}, \widehat{\delta}_{min}, \widehat{\delta}_{cyc}) \leftarrow$ maxDel$(n(e), j, k, (\chi'_1 + |\ell_j(e)|, \chi'_2 + |\ell_k(e)|),$
$\quad\quad\quad\quad (\widehat{\delta}_{max}, \widehat{\delta}_{min}, \widehat{\delta}_{cyc}))$
16 $\quad\quad \chi[q_1] \leftarrow \bot$
17 return $(\widehat{\delta}_{max}, \widehat{\delta}_{min}, \widehat{\delta}_{cyc})$

Next, we compile the first limit, $\delta_{\max}$, that will not be exceeded by a construction with bounded delay. In a match of two cycles this limit equals δ_{cyc}, and for any other match it is $\widehat{\delta}_{max} - \widehat{\delta}_{min}$. In a construction with bounded delay, the absolute value of the delay in $A^{(n)}$ does therefore not exceed $\delta_{max} = max(\delta_{cyc}\ ,\ \widehat{\delta}_{max} - \widehat{\delta}_{min})$ (Line 5).

Finally, we compile a second limit, $\delta_{\max 2}$, that allows us, in case of potentially unbounded delay, to construct a larger $A^{(n)}$ than $\delta_{\max}$ does. Unboundedness can only result from matching cycles in $A_1^{(n)}$. To obtain a larger $A^{(n)}$, with states whose delay exceeds $\delta_{\max}$, we have to unroll the cycles of $A_1^{(n)}$ further until we reach (at least) one more match between two cycles. Therefore, $\delta_{max2} = \delta_{max} + \delta_{cyc}$ (Line 6).

Construction: We start with a WMTA A whose alphabet and semiring equal those of A_1 and that is otherwise empty (Line 2). To each state q that will be created in A, we will assign two variables: $\nu[q] = q_1$ indicating the corresponding state q_1 in A_1, and $\xi[q] = (s, u)$ stating the leftover string s of tape j (yet unmatched in tape k) and the leftover string u of tape k (yet unmatched in tape j).

Then, we create an initial state i in A and push it onto the stack (Lines 4, 18–27). As long as the stack is not empty, we take states q from it and follow each of the outgoing transitions $e_1 \in E(q_1)$ of the corresponding state $q_1 = \nu[q]$ in A_1 (Lines 5–7). A transition e_1 in A_1 is represented as $e \in E(q)$ in A, with the same label and weight. To compile the leftover strings $\xi[q'] = (s', u')$ of its target $q' = n(e)$ in A, we concatenate the leftover strings $\xi[q] = (s, u)$ of its source $q = p(e)$ with the j-th and k-th component of its label, $\ell_j(e_1)$ and $\ell_k(e_1)$, and remove the longest common prefix of the resulting strings $s \cdot \ell_j(e_1)$ and $u \cdot \ell_k(e_1)$ (Lines 8, 14–17).

AUTOINTERSECT$(A_1, j, k) \rightarrow (A\,,\ boolean)$ **:**
1 $(\delta_{\max}, \delta_{\max 2}) \leftarrow$ GETMAXDELAYS(A_1, j, k)
2 $A \leftarrow \langle \Sigma_1, \emptyset, \bot, \emptyset, \emptyset, \mathcal{K}_1 \rangle$
3 $Stack \leftarrow \emptyset$
4 $i \leftarrow$ GETSTATE$(i_1, (\varepsilon, \varepsilon))$
5 while $Stack \neq \emptyset$ do
6     $q \leftarrow$ *pop*$(Stack) : \nu[q] = q_1\,, \xi[q] = (s, u)$
7   for $\forall e_1 \in E(q_1)$ do
8         $(s', u') \leftarrow$ CREATELEFTOVERSTRINGS$(s, \ell_j(e_1), u, \ell_k(e_1))$
9   if $(\ s' = \varepsilon \vee u' = \varepsilon\ )\ \wedge\ (\ |\delta(s', u')| \leq \delta_{\max 2}\ )$
10   then $q' \leftarrow$ GETSTATE$(n(e_1), (s', u'))$
11                 $E \leftarrow E \cup \{\ \langle q, \ell(e_1), w(e_1), q' \rangle\ \}$
12 $successful \leftarrow (\ \nexists q \in Q : |\delta(\xi[q])| > \delta_{\max}\ \wedge\ coreachable(q)\ )$
13 return $(\ A\,,\ successful\ )$

CREATELEFTOVERSTRINGS$(s_0, s_1, u_0, u_1) \rightarrow (s, u)$ **:**
14 $s \leftarrow s_0\ s_1$
15 $u \leftarrow u_0\ u_1$
16 $x \leftarrow$ *lcp*(s, u)
17 return $(x^{-1}\ s\,, x^{-1}\ u)$

GETSTATE$(q_1, (s', u')) \rightarrow q$ **:**
18 if $\exists q' \in Q : \nu[q'] = q_1\ \wedge\ \xi[q'] = (s', u')$
19   then $q \leftarrow q'$
20   else $Q \leftarrow Q \cup \{q\}$ [create new state]
21   if $s = \varepsilon\ \wedge\ u = \varepsilon$
22   then $\varrho(q) \leftarrow \varrho(q_1)$
23   else $\varrho(q) \leftarrow \bar{0}$
24         $\nu[q] \leftarrow q_1$
25         $\xi[q] \leftarrow (s', u')$
26   *push*$(Stack, q)$
27 return q

If both leftover strings s' and u' of q' are non-empty ($\neq \varepsilon$) then they are incompatible and the path that we are following is invalid. If either s' or u' is empty ($= \varepsilon$) then the current path is valid (at least up to this point) (Line 9). Only in this case and only if the delay between s' and u' does not exceed $\delta_{\max 2}$, we construct a transition e in A corresponding to e_1 in A_1 (Line 9, 11). If its target $q' = n(e)$ does not exist yet, it is created and pushed onto the stack (Lines 10, 18–27). The infinite unrolling of cycles is prevented by $\delta_{\max 2}$.

Verification: To see whether the construction was successful and whether $A^{(n)} = \mathcal{I}_{j,k}(A_1^{(n)})$, we have to check for the above defined conditions. Since all states of $A^{(n)}$ are reachable, it is sufficient to verify their delay and coreachability (Line 12) : $\nexists q : |\delta(q)| > \delta_{\max} \wedge q$ coreachable.

6.2.2 Examples

We illustrate the algorithm through the following three examples that stand each for a different class of WMTAs.

Example 1: The relation of the WMTA, $A_1^{(3)}$, of the first example is the infinite set of string tuples $\{\langle ab^k, xy^k z, a^k b\rangle | k \in \mathbb{N}\}$ (Figure 1). Only one of those tuples, namely $\langle ab, xyz, ab\rangle$, is in the relation of the auto-intersection, $A^{(3)} = \mathcal{I}_{1,3}(A_1^{(3)})$, because all other tuples contain different strings on tape 1 and 3. In the construction, an infinite unrolling of the cycle is prevented by the incompatibility of the leftover substrings in $\xi[3]$ and $\xi[4]$ respectively. The construction is successful.

The example is characterized by:

$$\delta_{\max} = \delta_{\max 2} = 1 \tag{49}$$

$$\mathcal{R}(A_1^{(3)}) = \{\langle ab^k, xy^k z, a^k b\rangle \mid k \in \mathbb{N}\} \tag{50}$$

$$\mathcal{I}_{1,3}(\mathcal{R}(A_1^{(3)})) = \mathcal{R}(A^{(3)}) = \{\langle ab^1, xy^1 z, a^1 b\rangle\} \tag{51}$$

$$\nexists q \in Q : |\delta(\xi[q])| > \delta_{\max} \Rightarrow \textit{successful} \Rightarrow \textit{rational } \mathcal{I}_{1,3}(\,) \tag{52}$$

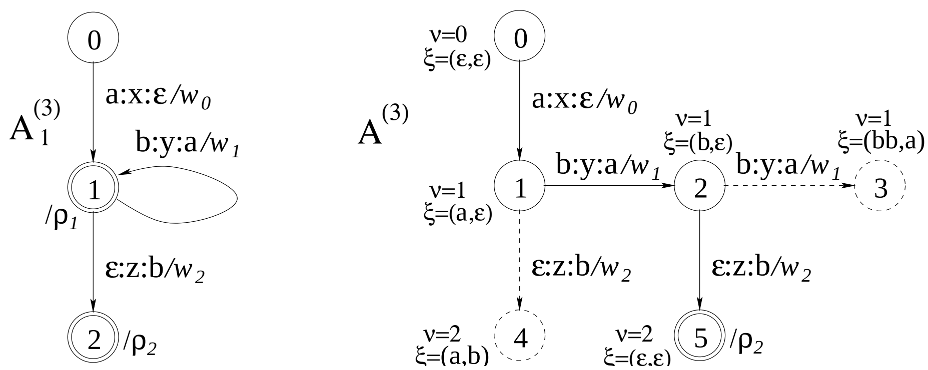

Figure 1: A WMTA $A_1^{(3)}$ and its successfully constructed auto-intersection $A^{(3)} = \mathcal{I}_{1,3}(A_1^{(3)})$. (Dashed parts are not constructed.)

Example 2: In the second example (Figure 2), the relation of $A_1^{(3)}$ is the infinite set of string tuples $\{\langle a^k, a, x^k y\rangle \mid k \in \mathbb{N}\}$. Only one of those tuples, namely $\langle a^1, a, x^1 y\rangle$, is in the relation of the auto-intersection $A^{(3)} = \mathcal{I}_{1,2}(A_1^{(3)})$. In the construction, an infinite unrolling of the cycle is prevented by the limit of delay $\delta_{\max 2}$. Although the result contains states with $\delta(\xi[q])| > \delta_{\max}$, none of them is coreachable (and would disappear if the result was pruned). The construction is successful.

The example is characterized by:

$$\delta_{\max} = 2 \tag{53}$$

$$\delta_{\max 2} = 3 \tag{54}$$

$$\mathcal{R}(A_1^{(3)}) = \{\langle a^k, a, x^k y\rangle \mid k \in \mathbb{N}\} \tag{55}$$

$$\mathcal{I}_{1,2}(\mathcal{R}(A_1^{(3)})) = \mathcal{R}(A^{(3)}) = \{\langle a^1, a, x^1 y\rangle\} \tag{56}$$

$$\nexists q \in Q : |\delta(\xi[q])| > \delta_{\max} \wedge \textit{coreachable}(q) \Rightarrow \textit{successful} \Rightarrow \textit{rational } \mathcal{I}_{1,2}(\,) \tag{57}$$

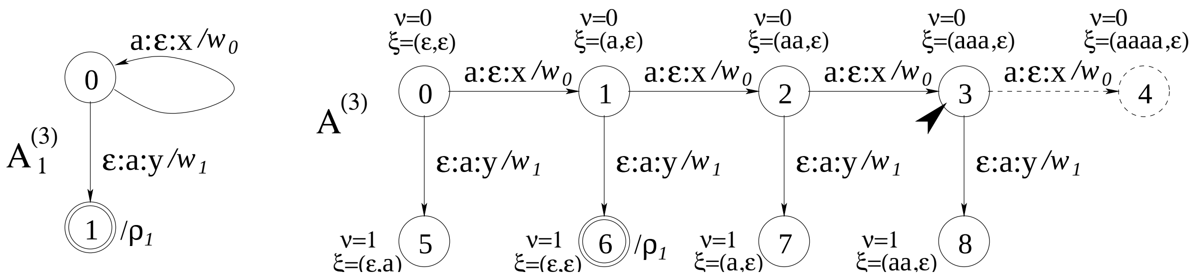

Figure 2: A WMTA $A_1^{(3)}$ and its successfully constructed auto-intersection $A^{(3)} = \mathcal{I}_{1,2}(A_1^{(3)})$. (Dashed parts are not constructed. States q marked with ➶ have $|\delta(\xi[q])| > \delta_{\max}$.)

Example 3: In the third example (Figure 3), the relation of $A_1^{(3)}$ is the infinite set of string tuples $\{\langle a^k a, aa^h, x^k yz^h\rangle \mid k,h \in \mathbb{N}\}$. The auto-intersection, $\mathcal{I}_{1,2}(A_1^{(3)})$, is not rational and has unbounded delay. Its complete construction would require an infinite unrolling of the cycles of $A_1^{(3)}$ and an infinite number of states in $A^{(3)}$ which is prevented by $\delta_{\max 2}$. The construction is not successful because the result contains coreachable states with $\delta(\xi[q])| > \delta_{\max}$.

The example is characterized by:

$$\delta_{\max} = 2 \tag{58}$$

$$\delta_{\max 2} = 3 \tag{59}$$

$$\mathcal{R}(A_1^{(3)}) = \{\langle a^k a, aa^h, x^k yz^h\rangle \mid k,h \in \mathbb{N}\} \tag{60}$$

$$\mathcal{I}_{1,2}(\mathcal{R}(A_1^{(3)})) = \{\langle a^k a, aa^k, x^k yz^k\rangle \mid k \in \mathbb{N}\} \tag{61}$$

$$\mathcal{I}_{1,2}(\mathcal{R}(A_1^{(3)})) \supset \mathcal{R}(A^{(3)}) = \{\langle a^k a, aa^k, x^k yz^k\rangle \mid k \in [\![0,3]\!]\} \tag{62}$$

$$\exists q \in Q : |\delta(\xi[q])| > \delta_{\max} \;\wedge\; \mathit{coreachable}(q) \Rightarrow \mathit{not\ successful} \tag{63}$$

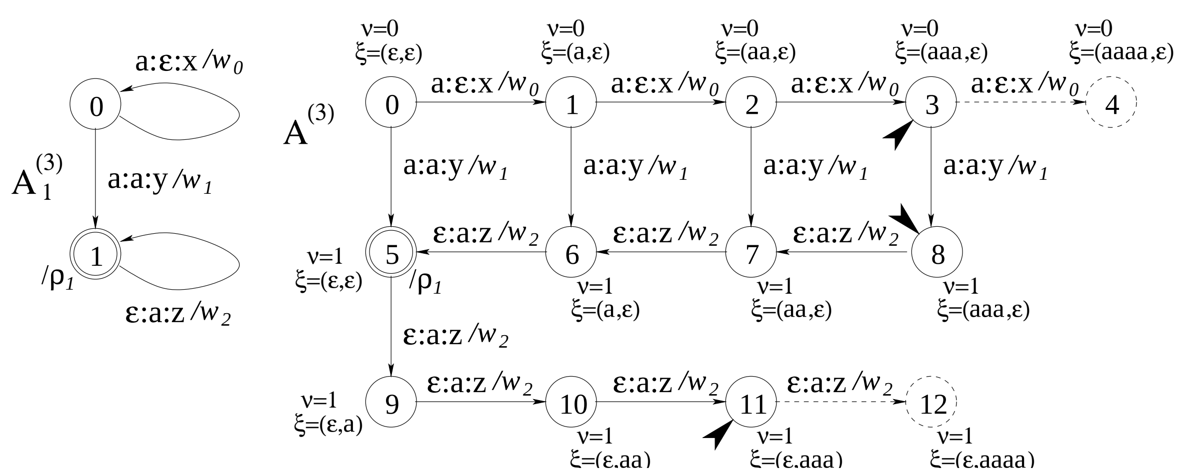

Figure 3: A WMTA $A_1^{(3)}$ and its partially constructed auto-intersection $A^{(3)} \subset \mathcal{I}_{1,2}(A_1^{(3)})$. (Dashed parts are not constructed. States q marked with ➶ have $|\delta(\xi[q])| > \delta_{\max}$.)

6.3 Single-Tape Intersection

We propose an algorithm that performs single-tape intersection of two WMTAs, $A_1^{(n)}$ and $A_2^{(m)}$, in one step. Instead of first building the cross-product, $A_1^{(n)} \times A_2^{(m)}$, and then deleting most of its paths by auto-intersection, $\mathcal{I}_{j,n+k}(\ )$, according to the above procedure (Eq. 37), the algorithm constructs only the useful part of the cross-product. It is very similar to classical composition of two transducers, and incorporates the idea of using an ε-filter in the composition of transducers containing ε-transitions (Mohri, Pereira, and Riley, 1998, Figure 10) that will be explained below. Instead of explicitly using an ε-filter, we simulate its behaviour in the algorithm. We will refer to the algorithm as IntersectCrossEps(A_1, A_2, j, k):

$$\textsc{IntersectCrossEps}(A_1, A_2, j, k) = \mathcal{I}_{j,n+k}(\ A_1^{(n)} \times A_2^{(m)}\ ) \tag{64}$$

$$A_1^{(n)} \underset{j,k}{\cap} A_2^{(m)} = \overline{\mathcal{P}}_{n+k}(\ \textsc{IntersectCross}(A_1, A_2, j, k)\ ) \tag{65}$$

The complementary projection, $\overline{\mathcal{P}}_{n+k}(\ )$, could be easily integrated into the algorithm in order to avoid an additional pass. We keep it apart because IntersectCrossEps( ) serves also as a building block of another algorithm where this projection must be postponed.

6.3.1 Mohri's ε-Filter

To compose two transducers, $A_1^{(2)}$ and $A_2^{(2)}$, containing ε-transitions, Mohri, Pereira, and Riley (1998, Figure 10) are using an ε-filter transducer. In their approach, $A_1^{(2)}$ and $A_2^{(2)}$ are pre-processed (Figure 4) : each ε on tape 2 of $A_1^{(2)}$ is replaced by an ε_1 and each ε on tape 1 of $A_2^{(2)}$ by an ε_2. In addition, a looping transition labeled with $\varepsilon : \phi_1$ is added to each state of $A_1^{(2)}$, and a loop labeled with $\phi_2 : \varepsilon$ to each state of $A_2^{(2)}$. The pre-processed transducers are then composed with the filter $A_\varepsilon^{(2)}$ in between: $A_1 \diamond A_\varepsilon \diamond A_2$.

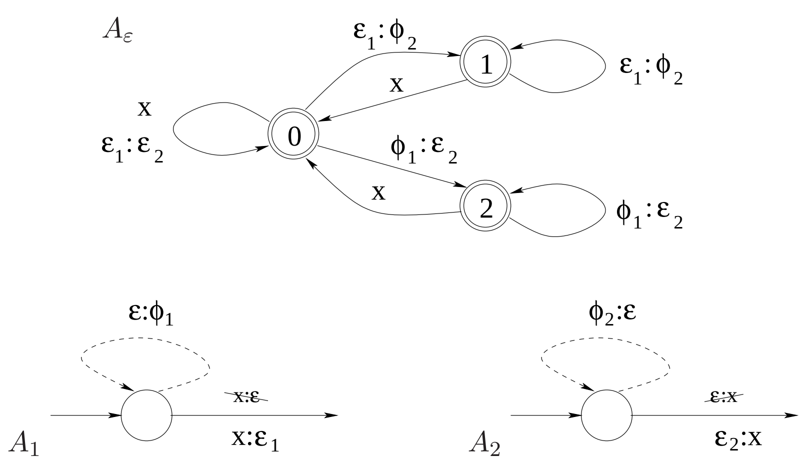

Figure 4: Mohri's ε-filter A_ε and two transducers, A_1 and A_2, pre-processed for filtered composition. $x = \neg\{\phi_1, \phi_2, \varepsilon_1, \varepsilon_2\}$. (For didactic reasons we are using slightly different labels than Mohri *et al*).

The filter controls how ε-transitions are composed along each pair of paths in A_1 and A_2 respectively. As long as there are equal symbols (ε or not) on the two paths, they are composed with each other and we stay in state 0 of A_ε. If we encounter a sequence of ε in A_1 but not in A_2, we move forward in A_1, stay in the same state in A_2, and in state 1 of A_ε. If we encounter a sequence of ε in A_2 but not in A_1, we move forward in A_2, stay in the same state in A_1, and in state 2 of A_ε.

6.3.2 Conditions

Our algorithm requires the semirings of the two WMTAs to be equal ($\mathcal{K}_1 = \mathcal{K}_2$) and commutative. All transitions must be labeled with n-tuples of strings not exceeding length 1 on the intersected tapes j of A_1 and k of A_2 which means no loss of generality: $\forall e_1 \in E_1 : |\ell_j(e_1)| \leq 1$; $\forall e_2 \in E_2 : |\ell_k(e_2)| \leq 1$

6.3.3 Algorithm

We start with a WMTA A whose alphabet is the union of the alphabets of A_1 and A_2, whose semiring equals those of A_1 and A_2, and that is otherwise empty (Line 1).

IntersectCrossEps$(A_1^{(n)}, A_2^{(m)}, j, k) \rightarrow A$:
1 $A \leftarrow \langle \Sigma_1 \cup \Sigma_2, \emptyset, \bot, \emptyset, \emptyset, \mathcal{K}_1 \rangle$
2 $Stack \leftarrow \emptyset$
3 $i \leftarrow$ getState$(i_1, i_2, 0)$
4 while $Stack \neq \emptyset$ do
5 $q \leftarrow$ *pop*$(Stack)$: $\vartheta[q] = (q_1, q_2, q_\varepsilon)$
6 for $\forall e_1 \in E(q_1)$ do
7 for $\forall e_2 \in E(q_2)$ do
8 if $\ell_j(e_1) = \ell_k(e_2) \ \wedge \ (q_\varepsilon = 0 \ \vee \ \ell_j(e_1) \neq \varepsilon)$
9 then $q' \leftarrow$ getState$(n(e_1), n(e_2), 0)$
10 $E \leftarrow E \cup \{ \langle q, \ \ell(e_1) : \ell(e_2), \ w(e_1) \otimes w(e_2), \ q' \rangle \}$
11 for $\forall e_1 \in E(q_1)$ do
12 if $\ell_j(e_1) = \varepsilon \ \wedge \ q_\varepsilon \in \{0, 1\}$
13 then $q' \leftarrow$ getState$(n(e_1), q_2, 1)$
14 $E \leftarrow E \cup \{ \langle q, \ \ell(e_1) : \varepsilon^{(m)}, \ w(e_1), \ q' \rangle \}$
15 for $\forall e_2 \in E(q_2)$ do
16 if $\ell_k(e_2) = \varepsilon \ \wedge \ q_\varepsilon \in \{0, 2\}$
17 then $q' \leftarrow$ getState$(q_1, n(e_2), 2)$
18 $E \leftarrow E \cup \{ \langle q, \ \varepsilon^{(n)} : \ell(e_2), \ w(e_2), \ q' \rangle \}$
19 return A

getState$(q_1, q_2, q_\varepsilon) \rightarrow q$:
20 if $\exists q' \in Q : \vartheta[q'] = (q_1, q_2, q_\varepsilon)$
21 then $q \leftarrow q'$
22 else $Q \leftarrow Q \cup \{q\}$ [create new state]
23 $\varrho(q) \leftarrow \varrho(q_1) \otimes \varrho(q_2)$
24 $\vartheta[q] \leftarrow (q_1, q_2, q_\varepsilon)$
25 *push*$(Stack, q)$
26 return q

First, we create the initial state i of A from the initial states of A_1, A_2, and A_ε, and push i onto the stack (Lines 3, 20–26). While the stack is not empty, we take states q from it and access the states q_1, q_2, and q_ε that are assigned to q through $\vartheta[q]$ (Lines 4, 5).

We intersect each outgoing transition e_1 of q_1 with each outgoing transition e_2 of q_2 (Lines 6, 7). This succeeds only if the j-th label component of e_1 equals the k-th label component of e_2, where j and k are the two intersected tapes of A_1 and A_2 respectively, and if the corresponding transition in A_ε has target 0 (Line 8). Only if it succeeds, we create a transition in A (Line 10) whose label results from pairing $\ell(e_1)$ with $\ell(e_2)$ and whose target q' corresponds with the triple of targets $(n(e_1), n(e_2), 0)$. If q' does not exist yet, it is created and pushed onto the stack (Lines 20–26).

Subsequently, we handle all ε-transitions in A_1 (Lines 11–14) and in A_2 (Lines 15–18). If we encounter an ε in A_1 and are in state 0 or 1 of A_ε, we have to move forward in A_1, stay in the same state in A_2, and go to state 1 in A_ε. Therefore we create a transition in A whose target corresponds to the triple $(n(e_1), q_2, 1)$ (Lines 11–14). The algorithm works similarly if and ε is encountered in A_2 (Lines 15–18).

To adapt this algorithm to non-weighted MTAs, one has to remove the weights from the Lines 10, 14, and 18, and replace Line 23 with: $Final(q) \leftarrow Final(q_1) \wedge Final(q_2)$.

6.4 Multi-Tape Intersection

We propose two alternative algorithms for the multi-tape intersection of two WMTAs, $A_1^{(n)}$ and $A_2^{(m)}$.

6.4.1 Conditions

Both algorithms work under the conditions of their underlying basic operations: The semirings of the two WMTAs must be equal ($\mathcal{K}_1 = \mathcal{K}_2$) and commutative. The second (more efficient algorithm) requires all transitions to be labeled with n-tuples of strings not exceeding length 1 on (at least) one pair of intersected tapes j_i of $A_1^{(n)}$ and k_i of $A_2^{(m)}$ which means no loss of generality: $\exists i \in [\![1, r]\!] : (\forall e_1 \in E_1 : |\ell_{j_i}(e_1)| \leq 1) \wedge (\forall e_2 \in E_2 : |\ell_{k_i}(e_2)| \leq 1)$

6.4.2 Algorithms

Our first algorithm, that we will refer to as INTERSECT1($A_1^{(n)}, A_2^{(m)}, j_1 \ldots j_r, k_1 \ldots k_r$), follows the exact procedure of multi-tape intersection (Eq. 37), using the algorithms for cross product, auto-intersection, and complementary projection.

INTERSECT1**(**$A_1^{(n)}, A_2^{(m)}, j_1 \ldots j_r, k_1 \ldots k_r$**)** $\rightarrow$ **(**$A\,,\ boolean$**) :**

  $successful \leftarrow true$
  $A \leftarrow$ CROSSPA($A_1^{(n)}, A_2^{(m)}$)
  for $\forall i \in [\![1, r]\!]$ do
    ($A\,,\ success$) $\leftarrow$ AUTOINTERSECT($A, j_i, n + k_i$)
    $successful \leftarrow successful \wedge success$
  $A \leftarrow \overline{\mathcal{P}}_{n+k_1, \ldots, n+k_r}(A)$
  return ($A\,,\ successful$)

The second (more efficient) algorithm, that we will call INTERSECT2($A_1^{(n)}, A_2^{(m)}, j_1 \ldots j_r, k_1 \ldots k_r$), uses first the above single-tape intersection algorithm to perform cross product and one auto-intersection in one single step (for intersecting tape j_1 with k_1), and then the auto-intersection algorithm (for intersecting all remaining tapes j_i with k_i, for $i > 1$).

INTERSECT2$(A_1^{(n)}, A_2^{(m)}, j_1 \dots j_r, k_1 \dots k_r) \rightarrow (A\,,\ boolean)$:

$successful \leftarrow true$

$A \leftarrow$ INTERSECTCROSSEPS$(A_1^{(n)}, A_2^{(m)}, j_1, k_1)$

for $\forall i \in [\![2, r]\!]$ do

  $(A\,,\ success) \leftarrow$ AUTOINTERSECT$(A, j_i, n + k_i)$

  $successful \leftarrow successful \wedge success$

$A \leftarrow \overline{\mathcal{P}}_{n+k_1,\, \dots\, ,n+k_r}(A)$

return $(A\,,\ successful)$

This second algorithm has been used to compile successfully the example of transducer intersection in Section 5.

7 Applications

Many applications of WMTAs and WMTA operations are possible, such as the morphological analysis of Semitic languages or the extraction of words from a bi-lingual dictionary that have equal meaning and similar form in the two languages (cognates).

We include only one example in this report, namely the preservation of intermediate results in transduction cascades, which actually stands for a large class of applications.

7.1 Preserving Intermediate Transduction Results

Transduction cascades have been extensively used in language and speech processing (Aït-Mokhtar and Chanod, 1997; Pereira and Riley, 1997; Kempe, 2000; Kumar and Byrne, 2003; Kempe et al., 2003, among many others).

In a (classical) weighted transduction cascade, $T_1^{(2)} \dots T_r^{(2)}$, a set of weighted input strings, encoded as a weighted acceptor, $L_0^{(1)}$, is composed with the first transducer, $T_1^{(2)}$, on its input tape (Figure 5). The output projection of this composition is the first intermediate result, $L_1^{(1)}$, of the cascade. It is further composed with the second transducer, $T_2^{(2)}$, which leads to the second intermediate result, $L_2^{(1)}$, etc. The output projection of the last transducer is the final result, $L_r^{(1)}$:

$$L_i^{(1)} = \mathcal{P}_2(\, L_{i-1}^{(1)} \diamond T_i^{(2)} \,) \qquad \text{for} \quad i \in [\![1, r]\!] \tag{66}$$

At any point in the cascade, previous results cannot be accessed. This holds also if the cascade is composed into a single transducer, $T^{(2)}$. None of the "incorporated" sub-relations in $T^{(2)}$ can refer to a sub-relation other than its immediate predecessor:

$$T^{(2)} = T_1^{(2)} \diamond \dots \diamond T_r^{(2)} \tag{67}$$

In a weighted transduction cascade, $A_1^{(n_1)} \dots A_r^{(n_r)}$, that uses WMTAs and multi-tape intersection, intermediate results can be preserved and used by all subsequent transductions. Suppose, we want to use

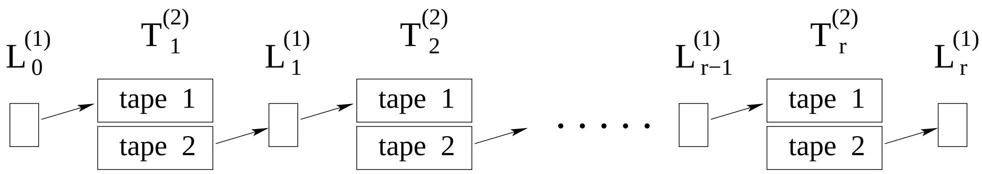

Figure 5: Weighted transduction cascade (classical)

the two previous results at each point in the cascade (except in the first transduction) which requires all intermediate results, $L_i^{(2)}$, to have two tapes (Figure 6) : The projection of the output-tape of the last WMTA is the final result, $L_r^{(1)}$:

$$L_1^{(2)} = L_0^{(1)} \underset{1,1}{\cap} A_1^{(2)} \tag{68}$$

$$L_i^{(2)} = \mathcal{P}_{2,3}(\ L_{i-1}^{(2)} \underset{\substack{1,1\\2,2}}{\cap} A_i^{(3)}\ ) \qquad \text{for} \quad i \in [\![2, r{-}1]\!] \tag{69}$$

$$L_r^{(1)} = \mathcal{P}_3(\ L_{r-1}^{(2)} \underset{\substack{1,1\\2,2}}{\cap} A_r^{(3)}\ ) \tag{70}$$

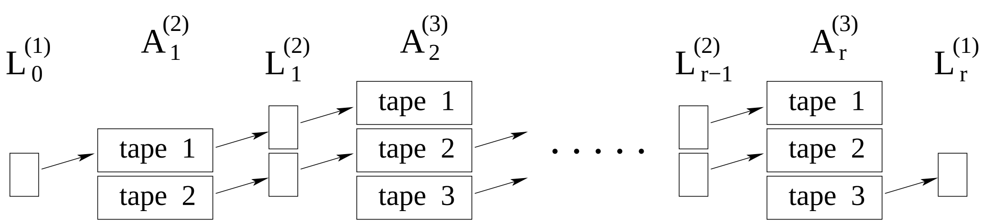

Figure 6: Weighted transduction cascade using multi-tape intersection (Example 1)

This augmented descriptive power is also available if the whole cascade is intersected into a single WMTA, $A^{(2)}$, although $A^{(2)}$ has only two tapes in our example. This can be achieved by intersecting iteratively the first i WMTAs until i reaches r :

$$A_{1\ldots i}^{(3)} = \mathcal{P}_{1,n-1,n}(\ A_{1\ldots i-1}^{(m)} \underset{\substack{n-1,1\\n,2}}{\cap} A_i^{(3)}\ ) \qquad \text{for} \quad i \in [\![2, r]\!]\,,\ m \in \{2,3\} \tag{71}$$

Each $A_{1\ldots i}^{(3)}$ contains all WMTAs from $A_1^{(2)}$ to $A_i^{(3)}$. The final result $A^{(2)}$ is built from $A_{1\ldots r}^{(3)}$:

$$A^{(2)} = \mathcal{P}_{1,n}(\ A_{1\ldots r}\ ) \tag{72}$$

Each (except the first) of the “incorporated” multi-tape sub-relations in $A^{(2)}$ will still refer to its two predecessors.

In our second example of a WMTA cascade, $A_1^{(n_1)} \ldots\ A_r^{(n_r)}$, each WMTA uses the output of its immediate predecessor, as in a classical cascade (Figure 7). In addition, the last WMTA uses the output of the first one:

$$L_1^{(2)} = L_0^{(1)} \underset{1,1}{\cap} A_1^{(2)} \tag{73}$$

$$L_i^{(2)} = \mathcal{P}_{1,3}(\ L_{i-1}^{(2)} \underset{2,1}{\cap} A_i^{(2)}\ ) \qquad \text{for} \quad i \in [\![2, r{-}1]\!] \tag{74}$$

$$L_r^{(1)} = \mathcal{P}_3(\ L_{r-1}^{(2)} \underset{\substack{1,1\\2,2}}{\cap} A_r^{(3)}\ ) \tag{75}$$

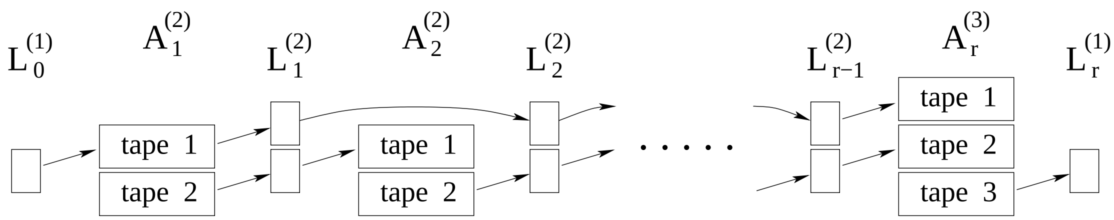

Figure 7: Weighted transduction cascade using WMTAs (Example 2)

As in the previous example, the cascade can be intersected into a single WMTA, $A^{(2)}$, that exceeds the power of a classical transducer cascade, although it has only two tapes:

$$A_{1\ldots i}^{(2)} = \mathcal{P}_{1,3}(\ A_{1\ldots i-1}^{(2)} \underset{2,1}{\cap} A_i^{(2)}\ ) \qquad \text{for} \quad i \in [\![2, r{-}1]\!] \tag{76}$$

$$A_{1\ldots r}^{(3)} = \mathcal{P}_{1,3}(\ A_{1\ldots r-1}^{(2)} \underset{\substack{1,1\\2,2}}{\cap} A_r^{(3)}\ ) \tag{77}$$

$$A^{(2)} = \mathcal{P}_{1,3}(\ A_{1\ldots r}^{(3)}\ ) \tag{78}$$

Acknowledgements

We wish to thank several anonymous reviewers.

References

Aït-Mokhtar, Salah and Jean-Pierre Chanod. 1997. Incremental finite-state parsing. In *Proc. 5th Int. Conf. ANLP*, pages 72–79, Washington, DC, USA.

Beesley, Kenneth R. and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Publications, Palo Alto, CA.

Birkhoff, Garrett and Thomas C. Bartee. 1970. *Modern Applied Algebra*. McGraw-Hill, New York, NY, USA.

Eilenberg, Samuel. 1974. *Automata, Languages, and Machines*, volume A. Academic Press, San Diego, CA, USA.

Elgot, Calvin C. and Jorge E. Mezei. 1965. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68.

Frougny, Christiane and Jacques Sakarovitch. 1993. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108(1):45–82.

Ganchev, Hristo, Stoyan Mihov, and Klaus U. Schulz. 2003. One-letter automata: How to reduce k tapes to one. CIS-Bericht 03-133, Centrum für Informations- und Sprachverarbeitung, Universität München.

Harju, Tero and Juhani Karhumäki. 1991. The equivalence problem of multitape finite automata. *Theoretical Computer Science*, 78(2):347–355.

Kaplan, Ronald M. and Martin Kay. 1981. Phonological rules and finite state transducers. In *Winter Meeting of the Linguistic Society of America*, New York, NY, USA.

Kaplan, Ronald M. and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Karttunen, Lauri, Jean-Pierre Chanod, Greg Grefenstette, and Anne Schiller. 1997. Regular expressions for language engineering. *Journal of Natural Language Engineering*, 2(4):307–330.

Kay, Martin. 1987. Nonconcatenative finite-state morphology. In *Proc. 3rd Int. Conf. EACL*, pages 2–10, Copenhagen, Denmark.

Kempe, André. 2000. Reduction of intermediate alphabets in finite-state transducer cascades. In *Proc. 7th Conf. TALN*, pages 207–215, Lausanne, Switzerland, October. ATALA.

Kempe, André, Christof Baeijs, Tamás Gaál, Franck Guingne, and Florent Nicart. 2003. WFSC – A new weighted finite state compiler. In O. H. Ibarra and Z. Dang, editors, *Proc. 8th Int. Conf. CIAA*, volume 2759 of *Lecture Notes in Computer Science*, pages 108–119, Santa Barbara, CA, USA. Springer Verlag, Berlin, Germany.

Kiraz, George Anton. 1997. Linearization of nonlinear lexical representations. In John Coleman, editor, *Proc. 3rd Meeting, ACL Special Interest Group in Computational Phonology*, Madrid, Spain.

Kiraz, George Anton and Edmund Grimley-Evans. 1998. Multi-tape automata for speech and language systems: A Prolog implementation. In D. Woods and S. Yu, editors, *Automata Implementation*, volume 1436 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany, pages 87–103.

Koskenniemi, Kimmo, Pasi Tapanainen, and Atro Voutilainen. 1992. Compiling and using finite-state syntactic rules. In *Proc. 16th Int. Conf. COLING*, volume 1, pages 156–162, Nantes, France.

Kuich, Werner and Arto Salomaa. 1986. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer Verlag, Berlin, Germany.

Kumar, Shankar and William Byrne. 2003. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proc. Int. Conf. HLT-NAACL*, pages 63–70, Edmonton, Canada.

Mohri, Mehryar. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–312.

Mohri, Mehryar. 2002. Generic epsilon-removal and input epsilon-normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143.

Mohri, Mehryar. 2003. Edit-distance of weighted automata. In *Proc. 7th Int. Conf. CIAA (2002)*, volume 2608 of *Lecture Notes in Computer Science*, pages 1–23, Tours, France. Springer Verlag, Berlin, Germany.

Mohri, Mehryar, Fernando C. N. Pereira, and Michael Riley. 1998. A rational design for a weighted finite-state transducer library. *Lecture Notes in Computer Science*, 1436:144–158.

Pereira, Fernando C. N. and Michael D. Riley. 1997. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*. MIT Press, Cambridge, MA, USA, pages 431–453.

Rabin, Michael O. and Dana Scott. 1959. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125.

Roche, Emmanuel and Yves Schabes. 1997. *Finite-State Language Processing*. MIT Press, Cambridge, MA, USA.

Sproat, Richard. 1992. *Morphology and Computation*. MIT Press, Cambridge, MA, USA.

van Noord, Gertjan. 1997. FSA Utilities: A toolbox to manipulate finite-state automata. In D. Raymond, D. Woods, and S. Yu, editors, *Automata Implementation*, volume 1260 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany.