WFSM Auto-Intersection and Join Algorithms

A. Kempe[1], J.-M. Champarnaud[2], F. Guingne[1,3], F. Nicart[1,3]

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

[2] PSI Laboratory (Université de Rouen, CNRS)
76821 Mont-Saint-Aignan – France
Jean-Marc.Champarnaud@univ-rouen.fr – http://www.univ-rouen.fr/psi/

[3] LIFAR Laboratory (Université de Rouen)
76821 Mont-Saint-Aignan – France
{Franck.Guingne,Florent.Nicart}@univ-rouen.fr
http://www.univ-rouen.fr/LIFAR/

Abstract. The join of two n-ary string relations is a main operation regarding to applications. n-Ary rational string relations are realized by weighted finite-state machines with n tapes. We provide an algorithm that computes the join of two machines via a more simple operation, the auto-intersection. The two operations generally do not preserve rationality. A delay-based algorithm is described for the case of a single tape pair, as well as the class of auto-intersections that it handles. It is generalized to multiple tape pairs and some enhancements are discussed.

1 Introduction

Multi-tape finite-state machines (FSMs) [16, 3, 7, 5, 6] are a natural generalization of the familiar finite-state acceptors (one tape) and transducers (two tapes). The n-ary relation defined by a (weighted) FSM is a (weighted) *rational* relation. Finite relations are of particular interest since they can be viewed as relational databases.[4] Multi-tape machines have been used in the morphological analysis of Semitic languages, to synchronize the vowels, consonants, and templatic pattern into a surface form [7, 12]. The operation of *join on multiple pairs of tapes*, that is similar to *natural join* of databases, is a crucial operation in many practical applications. In this paper, we focus on its computation through more basic operations such as the auto-intersection. The rationality of a join relation is generally undecidable, and so is the rationality of an auto-intersection relation [9]. In the case of a single pair of tapes, a class Θ of triples $\langle A, i, j\rangle$ can be defined so that the auto-intersection of the machine A w.r.t. tapes i and j can be computed by a delay-based algorithm. This algorithm is generalized to the case of multiple pairs of tapes, leading to a basic algorithm for computing the

[4] The connection to databases and associated notation was pointed out by J. Eisner in the joint work [9]. We thank him for allowing us to re-use this material.

join of two machines and to an improved version based on the notion of filtering and on the operation of equi-join on a single pair of tapes.

Weighted n-ary relations and their machines are introduced in Section 2. Join and auto-intersection operations are presented in Section 3. A basic algorithm for computing the join of two machines and the embedded auto-intersection algorithm are described in Section 4. We conclude by some enhancements.

2 Definitions

We recall some definitions about n-ary weighted relations and their machines, following the usual definitions for multi-tape automata [3, 1], with semiring weights added just as for acceptors and transducers [13, 15]. For more details see [9].

Weighted n-ary relations: A weighted n-ary relation is a function from $(\Sigma^*)^n$ to $\mathbb{K}$, for a given finite alphabet Σ and a given weight semiring $\mathcal{K} = \langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$. Such a relation assigns a weight to any n-tuple of strings. A weight of $\bar{0}$ means that the tuple is not in the relation.[5] We are especially interested in *rational* (or *regular*) n-ary relations, i.e. relations that can be encoded by n-tape weighted finite-state machines, that we now define. By convention, the names of objects containing n-tuples of strings include a superscript $^{(n)}$.

Multi-tape weighted finite-state machines: An *n-tape weighted finite-state machine* (WFSM or n-WFSM) $A^{(n)}$ is defined by a six-tuple $A^{(n)} = \langle \Sigma, Q, \mathcal{K}, E^{(n)}, \lambda, \varrho \rangle$, with Σ being a finite alphabet, Q a finite set of states, $\mathcal{K} = \langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$ the semiring of weights, $E^{(n)} \subseteq (Q \times (\Sigma^*)^n \times \mathbb{K} \times Q)$ a finite set of weighted n-tape transitions, $\lambda : Q \to \mathbb{K}$ a function that assigns initial weights to states, and $\varrho : Q \to \mathbb{K}$ a function that assigns final weights to states. Any transition $e^{(n)} \in E^{(n)}$ has the form $e^{(n)} = \langle p, \ell^{(n)}, w, n \rangle$. We refer to these four components as the transition's source state $p(e^{(n)}) \in Q$, its label $\ell(e^{(n)}) \in (\Sigma^*)^n$, its weight $w(e^{(n)}) \in \mathbb{K}$, and its target state $n(e^{(n)}) \in Q$. We refer by $E(q)$ to the set of out-going transitions of a state $q \in Q$ (with $E(q) \subseteq E^{(n)}$).

A *path* $\gamma^{(n)}$ of length $k \geq 0$ is a sequence of transitions $e_1^{(n)} e_2^{(n)} \cdots e_k^{(n)}$ such that $n(e_i^{(n)}) = p(e_{i+1}^{(n)})$ for all $i \in [\![1, k-1]\!]$. The label of a path is the element-wise concatenation of the labels of its transitions. The weight of a path $\gamma^{(n)}$ from q to q' is the product of the initial weight of q, the weights of the successive transitions and the final weight of q'. The path is said to be *successful*, and to *accept* its label, if $w(\gamma^{(n)}) \neq \bar{0}$. We denote by $\Gamma_{A^{(n)}}$ the set of all successful paths of $A^{(n)}$, and by $\Gamma_{A^{(n)}}(s^{(n)})$ the set of successful paths that accept the n-tuple of strings $s^{(n)}$. The machine $A^{(n)}$ defines a weighted n-ary relation $\mathcal{R}(A^{(n)}) : (\Sigma^*)^n \to \mathbb{K}$ that assigns to each n-tuple $s^{(n)}$ the total weight of all paths accepting it.

[5] It is convenient to define the *support* of an arbitrary weighted relation $\mathcal{R}^{(n)}$, as being the set of tuples to which the relation gives non-$\bar{0}$ weight.

3 Operations

We now describe some central operations on n-ary weighted relations and their n-WFSMs [11]. The auto-intersection operation is introduced, with the aim of simplifying the computation of the join operation. Our notation is inspired by relational databases. Mathematical details can be found in [9].

Simple Operations: Any n-ary weighted rational relation can be constructed by combining the basic rational operations of *union*, *concatenation* and *closure*. Rational operations can be implemented by simple constructions on the corresponding nondeterministic n-tape WFSMs [17]. These n-tape constructions and their semiring-weighted versions are exactly the same as for acceptors and transducers, since they are indifferent to the n-tuple transition labels.

The *projection* operator $\pi_{\langle j_1,\dots j_m\rangle}$, with $j_1,\dots j_m \in [\![1,n]\!]$, maps an n-ary relation to an m-ary one by retaining in each tuple components specified by the indices $j_1,\dots j_m$ and placing them in the specified order. Indices may occur in any order, possibly with repeats. Thus the tapes can be permuted or duplicated: $\pi_{\langle 2,1\rangle}$ inverts a 2-ary relation. The *complementary projection* operator $\overline{\pi}_{\{j_1,\dots j_m\}}$ removes the tapes $j_1,\dots j_m$ and preserves the order of other tapes.

Join operation: Our *join* operator differs from database join in that database columns are named, whereas our tapes are numbered. Tapes being explicitly selected by number, join is neither associative nor commutative.

For any distinct $i_1,\dots i_r \in [\![1,n]\!]$ and any distinct $j_1,\dots j_r \in [\![1,m]\!]$, the *join* operator $\bowtie_{\{i_1=j_1,\dots i_r=j_r\}}$ combines an n-ary and an m-ary relation into an $(n+m-r)$-ary relation defined as follows:[6]

$$\left(\mathcal{R}_1^{(n)} \bowtie_{\{i_1=j_1,\dots i_r=j_r\}} \mathcal{R}_2^{(m)}\right)(\langle u_1,\dots u_n, s_1,\dots s_{m-r}\rangle) =_{\text{def}} \mathcal{R}_1^{(n)}(u^{(n)}) \otimes \mathcal{R}_2^{(m)}(v^{(m)}) \quad (1)$$

$v^{(m)}$ being the unique tuple s. t. $\overline{\pi}_{\{j_1,\dots j_r\}}(v^{(m)}) = s^{(m-r)}$ and $(\forall k \in [\![1,r]\!])\ v_{j_k} = u_{i_k}$.

The *intersection* of two n-ary relations is the n-ary relation defined by the join operator $\bowtie_{\{1=1,2=2,\dots n=n\}}$. A join on a single pair (resp. multiple pairs) of tapes is said to be a *single-pair* (resp. *multi-pair*) one. Examples of single-pair join are the join $\bowtie_{\{1=1\}}$ (the intersection of two acceptors) and the join $\bowtie_{\{2=1\}}$ that can be used to express transducer composition.

A lot of practical applications could not be performed without the multi-tape join operation, for example: multi-tape transduction (mapping n-tuples to m-tuples of strings), probabilistic normalization of n-WFSMs conditioned on multiple tapes,[7] or searching for cognates [8].

Unfortunately, rational relations are *not* closed under arbitrary joins [9]. For example, transducers are not closed under intersection [16]. The join operation

[6] For example the tuples $\langle abc, def, \epsilon\rangle$ and $\langle def, ghi, \epsilon, jkl\rangle$ combine in the join $\bowtie_{\{2=1,3=3\}}$ and yield the tuple $\langle abc, def, \epsilon, ghi, jkl\rangle$, with a weight equal to the product of their weights.

[7] This can be obtained by a straightforward generalization of J. Eisner's algorithm for probabilistic normalization of transducers conditioned on one tape [2].

is, however, so useful that it is helpful to have a partial algorithm: hence our motivation for studying auto-intersection.

Auto-Intersection: For any distinct $i_1, j_1, \ldots\ i_r, j_r \in [\![1, n]\!]$, we define an *auto-intersection* operator $\sigma_{\{i_1=j_1, i_2=j_2, \ldots i_r=j_r\}}$. It maps a relation $\mathcal{R}^{(n)}$ to a subset of that relation, preserving tuples $s^{(n)}$ whose elements are equal in pairs as specified, but removing other tuples from the support of the relation:[8]

$$\left(\sigma_{\{i_1=j_1,\ldots i_r=j_r\}}(\mathcal{R}^{(n)})\right)(\langle s_1, \ldots s_n\rangle) =_{\text{def}} \begin{cases} \mathcal{R}^{(n)}(\langle s_1, \ldots s_n\rangle) & \text{if } (\forall k \in [\![1, r]\!]) s_{i_k} = s_{j_k} \\ \bar{0} & \text{otherwise} \end{cases} \quad (2)$$

Auto-intersecting a relation is different from joining it with its own projections. For example, $\sigma_{\{1=2\}}(\mathcal{R}^{(2)})$ is supported by tuples of the form $\langle w, w\rangle \in \mathcal{R}^{(2)}$. By contrast, $\mathcal{R}^{(2)} \bowtie_{\{1=1\}} \left(\pi_{\langle 2\rangle}(\mathcal{R}^{(2)})\right)$ is supported by tuples $\langle w, x\rangle \in \mathcal{R}^{(2)}$ such that w can also appear on tape 2 of $\mathcal{R}^{(2)}$ (but not necessarily paired with a copy of w on tape 1).[9]

Actually, join and auto-intersection are related by the following equalities:

$$\mathcal{R}_1^{(n)} \bowtie_{\{i_1=j_1,\ldots i_r=j_r\}} \mathcal{R}_2^{(m)} \;=\; \overline{\pi}_{\{n+j_1,\ldots n+j_r\}} \left(\sigma_{\{i_1=n+j_1,\ldots i_r=n+j_r\}} (\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)}) \right) \quad (3)$$

$$\sigma_{\{i_1=j_1,\ldots i_r=j_r\}}(\mathcal{R}^{(n)}) \;=\; \mathcal{R}^{(n)} \bowtie_{\{i_1=1, j_1=2, \ldots i_r=2r-1, j_r=2r\}} (\pi_{\langle 1,1\rangle}(\Sigma^*))^r \quad (4)$$

Thus, for any class of difficult join instances whose results are non-rational or have undecidable properties [9], there is a corresponding class of difficult auto-intersection instances, and vice-versa. Conversely, a partial solution to one problem would yield a partial solution to the other.

An auto-intersection on a single pair (resp. multiple pairs) of tapes is said to be a *single-pair* (resp. *multi-pair*) one. It may be wise to compute $\sigma_{\{i_1=j_1,\ldots i_r=j_r\}}$ all at once rather than one tape pair at a time, since a sequence of single-pair auto-intersections such as $\sigma_{\{i_r=j_r\}}(\cdots(\sigma_{\{i_1=j_1\}})\cdots)$ could fail due to non-rational intermediate results, even if the final result is rational.[10]

4 Join via auto-intersection: a first construction

Following (3), a multi-pair join can be computed via a multi-pair auto-intersection. A first version of such a join algorithm is presented in this section. The embedded multi-pair auto-intersection algorithm is a generalization of the single-pair one, that has been proved to work for a specific class of auto-intersections [10].

[8] The requirement that the $2r$ indices be distinct mirrors the similar requirement on join and is needed in (4). But it can be evaded by duplicating tapes.

[9] Applying $\sigma_{\{1=2\}}$ to $\{\langle a, b\rangle, \langle b, a\rangle\}$ yields the empty relation, whereas joining it with its own projection (either $\bowtie_{\{1=1\}} \pi_{\langle 2\rangle}$ or $\bowtie_{\{2=1\}} \pi_{\langle 1\rangle}$) does not change the relation.

[10] Applying $\sigma_{\{2=3,4=5\}}$ to $\{\langle a^i b^j, c^i, c^j, x, y\rangle \mid i, j \in \mathbb{N}\}$ yields the empty relation, while applying $\sigma_{\{2=3\}}$ yields the non-rational relation $\{\langle a^i b^i, c^i, c^i, x, y\rangle \mid i \in \mathbb{N}\}$.

4.1 Multi-pair join: a basic algorithm

The Algorithm JOIN1 attempts to construct the join of two WFSMs, $A_1^{(n)}$ and $A_2^{(m)}$, on multiple pairs of tapes specified by a set of constraints $T = \{t_1{=}(i_1{=}j_1), \dots t_r{=}(i_r{=}j_r)\}$. We write $\bowtie_T$ instead of $\bowtie_{\{i_1=j_1,\dots i_r=j_r\}}$.

JOIN1$(A_1^{(n)}, A_2^{(m)}, T) \to A^{(n+m-r)}$: $\quad [\, T = \{t{=}(i{=}j)\};\ |T| = r;\ A^{(n+m-r)} = A_1^{(n)} \bowtie_T A_2^{(m)} \,]$

1 $\quad A^{(n+m)} \leftarrow A_1^{(n)} \times A_2^{(m)}$
2 $\quad$ if $\quad |T| \neq 0$
3 $\quad\quad$ then
4 $\quad\quad\quad A^{(n+m)} \leftarrow$ AUTOINTERSECTION$(A^{(n+m)}, T)$
5 $\quad\quad\quad$ if $\quad A^{(n+m)} = \perp \qquad$ [error code]
6 $\quad\quad\quad\quad$ then return $\perp$
7 $\quad\quad\quad A^{(n+m-r)} \leftarrow \overline{\pi}_{\{n+j_h \mid t_h=(i_h=j_h)\in T\}}(A^{(n+m)})$
8 $\quad$ return $A^{(n+m-r)}$

We compile first the cross-product $A^{(n+m)}$ of $A_1^{(n)}$ and $A_2^{(m)}$. If T is empty, we simply return the crossproduct $A^{(n+m)}$ (Line 2). Otherwise we compile the auto-intersection of $A^{(n+m)}$ for all specified pairs of tapes (Line 4). The auto-intersection may fail and return an error code, in which case the join algorithm must return an error code as well (Lines 5, 6).

4.2 A class of rational single-pair auto-intersections

We now introduce a single-pair auto-intersection algorithm and the class of bounded delay auto-intersections that this algorithm can handle. For a detailed exposure see [10].

Although due to Post's Correspondence Problem there exists no fully general algorithm of auto-intersection [9], $A^{(n)} = \sigma_{\{i=j\}}(A_1^{(n)})$ can be compiled for a class of triples $\langle A_1^{(n)}, i, j\rangle$ whose definition is based on the notion of *delay* [4, 14], i.e., the difference of length of two strings of an n-tuple: $\delta_{\langle i,j\rangle}(s^{(n)}) = |s_i| - |s_j|$ (with $i, j \in [\![1, n]\!]$). The delay of a path $\gamma = \gamma_1\gamma_2\cdots\gamma_r$, or of any of its factors γ_h, results from its respective labels on tapes i and j: $\delta_{\langle i,j\rangle}(\gamma) = |\ell_i(\gamma)| - |\ell_j(\gamma)|$. We call the delay *bounded* if its absolute value does not exceed some limit. A path has bounded delay if all its prefixes have bounded delay,[11] and an n-WFSM has bounded delay if all its successful paths have bounded delay.

We construct $A^{(n)}$ without creating invalid paths with $\ell_i(\gamma) \neq \ell_j(\gamma)$, which is equivalent to creating them with $w(\gamma) = \bar{0}$. Thus, all paths of $A^{(n)}$ have a delay equal to 0 : Let Γ^0 be the set of accepting paths of $A_1^{(n)}$ with a 0-delay. Then it

[11] Any finite path has bounded delay (since its label is of finite length). An infinite path (traversing cycles) may have bounded or unbounded delay. For example, the delay of a path labeled with $(\langle ab, \varepsilon\rangle\langle \varepsilon, xz\rangle)^h$ is bounded by 2 for any h, whereas that of a path labeled with $\langle ab, \varepsilon\rangle^h\langle \varepsilon, xz\rangle^h$ is unbounded for $h \longrightarrow \infty$.

holds: $\Gamma_{A^{(n)}} \subseteq \Gamma^0 \subseteq \Gamma_{A_1^{(n)}}$. The sum of the delays of the factors of a path is equal to its delay, and it holds: $\forall \gamma = \gamma_1\gamma_2\cdots\gamma_r \in \Gamma^0,\ \delta_{\langle i,j\rangle}(\gamma) = \sum_{h=1}^{r} \delta_{\langle i,j\rangle}(\gamma_h) = 0$.

Let us traverse $A_1^{(n)}$ in-depth,[12] both left-to-right and right-to-left, and memorize the global maxima $\hat{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})$ and $\hat{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})$, and global minima $\check{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})$ and $\check{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})$ of the delay on any path. Let us then observe the delay along a path $\gamma \in \Gamma^0$: It would begin and end with $\delta_{\langle i,j\rangle} = 0$, and have a global maximum $\hat{\delta}_{\langle i,j\rangle}(\gamma)$ and a global minimum $\check{\delta}_{\langle i,j\rangle}(\gamma)$.

Proposition 1. *Let Θ be the class of all the triples $\langle A_1^{(n)}, i, j\rangle$ such that $A_1^{(n)}$ does not contain a path traversing both a cycle with positive delay and a cycle with negative delay (w.r.t. tapes i and j). Then for all paths $\gamma \in \Gamma_{A^{(n)}}$ of $A^{(n)} = \sigma_{\{i=j\}}(A_1^{(n)})$, the delay is bounded by*

$$\delta^{\mathsf{max}}_{\langle i,j\rangle} \;=\; \max(\, |\hat{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})|\,,\ |\hat{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})|\,,\ |\check{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})|\,,\ |\check{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})|\,) \quad (5)$$

Proof. If a path $\gamma \in \Gamma^0$ has only cycles with positive delay, traversing a cycle raises the delays in γ's suffix. These cycles have, however, no impact on the delays in the in-depth traversals, where cycles are not traversed. Therefore $(\ \check{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)}) \leq \check{\delta}_{\langle i,j\rangle}(\gamma) \leq 0\ )$ and $(\ \hat{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)}) \geq \hat{\delta}_{\langle i,j\rangle}(\gamma) \geq 0\ )$ which means

$$\forall \gamma \in \Gamma^0,\ \ \max(\, |\hat{\delta}_{\langle i,j\rangle}(\gamma)|, |\check{\delta}_{\langle i,j\rangle}(\gamma)|\,) \;\leq\; \max(\, |\check{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})|, |\hat{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})|\,) \quad (6)$$

This still holds if we also admit cycles with 0-delay on γ, since traversing them has no impact on the delays of γ's suffix. If all cycles of γ had negative or 0-delay instead, we would obtain

$$\forall \gamma \in \Gamma^0,\ \ \max(\, |\hat{\delta}_{\langle i,j\rangle}(\gamma)|, |\check{\delta}_{\langle i,j\rangle}(\gamma)|\,) \;\leq\; \max(\, |\check{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})|, |\hat{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})|\,) \quad (7)$$

Since $\Gamma_{A^{(n)}} \subseteq \Gamma^0$, (6) (7) and Proposition 1 hold for all paths $\gamma \in \Gamma_{A^{(n)}}$. ■

Joining $A_1^{(n)}$ beforehand with its own (neutrally weighted) projections yields a superset of $A^{(n)}$: $\mathsf{support}((A_1^{(n)} \bowtie_{\{i=1\}} \pi_{\langle j\rangle}(A_1^{(n)})) \bowtie_{\{j=1\}} \pi_{\langle i\rangle}(A_1^{(n)})) \supseteq \mathsf{support}(A^{(n)})$. The triple $\langle A_1^{(n)}, i, j\rangle$ is placed into Θ, as soon as this operation removes from $A_1^{(n)}$ all cycles in conflict with Θ. This method is referred as *filtering* and performed prior to any auto-intersection (it is the function FilterTapePairs of the Algorithm AutoIntersection in Section 4.4). Based on Proposition 1, an algorithm can be designed to compute $\sigma_{\{i=j\}}(A_1^{(n)})$ as far as $\langle A_1^{(n)}, i, j\rangle \in \Theta$. This algorithm is now described in a more general case.

[12] We optionally trim the automaton to restrict it to accepting paths. Then, to find (for example) $\hat{\delta}^{LR}_{\langle i,j\rangle}$, we exhaustively explore all acyclic paths from the start state, and record the maximum delay on any path prefix. This takes exponential time in general, which is unavoidable since the longest-acyclic-path problem is NP-complete.

4.3 Multi-pair auto-intersection: a basic construction

Our construction bears resemblance to known transducer synchronization procedures [4, 14]. However the algorithm of Frougny and Sakarovitch [4] is based on a $\mathbb{K}$-covering of the transducer and it works only for non-empty input labels whereas our single-pair auto-intersection algorithm supports unrestricted labeling. Our algorithm is based on a general reachability-driven construction, as it is the case for the synchronization algorithm of Mohri [14]. But the labeling of the transitions is quite different since our algorithm performs a copy of the original labeling, and we also construct only such paths whose delay does not exceed some limit that we are able to determine.

We now address the case of a multi-pair auto-intersection $\sigma_{\{i_1=j_1,\dots i_r=j_r\}}$ such that for all $h \in [\![1, r]\!]$, $\langle A_1^{(n)}, i_h, j_h\rangle \in \Theta$. As an example, we consider the WFSM $A_1^{(4)}$ in Figure 1a and the auto-intersection $\sigma_{\{1=2,3=4\}}(A_1^{(4)})$, with $\langle A_1^{(4)}, 1, 2\rangle \in \Theta$ and $\langle A_1^{(4)}, 3, 4\rangle \in \Theta$; the associated delay limits are $\delta^{\max}_{\langle 1,2\rangle} = 1$ and $\delta^{\max}_{\langle 3,4\rangle} = 2$. The support $(a{:}a{:}dc{:}cd \cup a{:}\varepsilon{:}c{:}\varepsilon)^*\ (ba{:}ab{:}c{:}\varepsilon)^*\ \varepsilon{:}a{:}\varepsilon{:}cc$ of $A_1^{(4)}$ is equal to the set $\{\ \langle a^{i+j}(ba)^h, a^i(ab)^h a, ([dc]^i \sqcup\!\sqcup c^j)c^h, (cd)^i c^2\rangle \mid i, j, h \in \mathbb{N}\ \}$.[13]

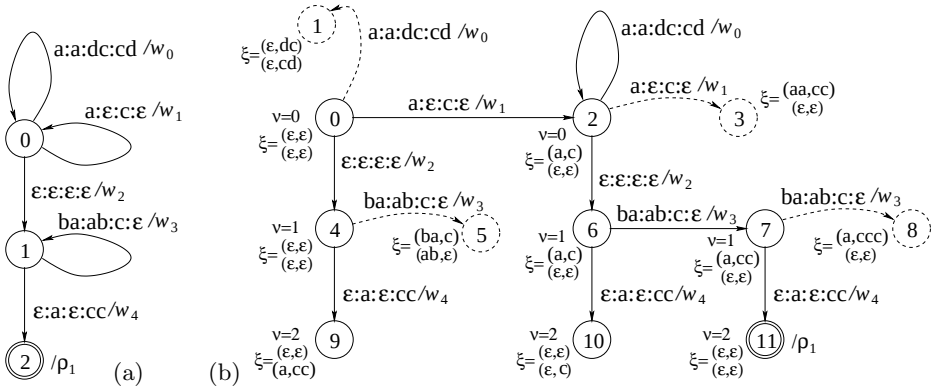

Fig. 1. (a) A WFSM $A_1^{(4)}$ and (b) its auto-intersection $A^{(4)} = \sigma_{\{1=2,3=4\}}(A_1^{(4)})$ (dashed parts are not constructed).

We construct simultaneously the two auto-intersections $\sigma_{\{1=2\}}$ and $\sigma_{\{3=4\}}$. We copy states and transitions one by one from $A_1^{(4)}$ (Figure 1a) to $A^{(4)}$ (Figure 1b), starting with the initial state $q_1 = 0$. We assign to each state q of $A^{(4)}$ two variables: $\nu[q] = q_1$ is the corresponding state q_1 of $A_1^{(4)}$, and $\xi[q] = (s^{(r)}, u^{(r)})$ expresses the leftover string tuple $s^{(r)}$ (resp. $u^{(r)}$) from the tapes $\langle i_1, \dots i_r\rangle$ (resp. $\langle j_1, \dots j_r\rangle$), yet unmatched on the tapes $\langle j_1, \dots j_r\rangle$ (resp. $\langle i_1, \dots i_r\rangle$). In particular, we have: $\nu[0] = 0$ and $\xi[0] = (\langle\varepsilon, \varepsilon\rangle, \langle\varepsilon, \varepsilon\rangle)$.

[13] A square-bracketted string cannot be split by shuffle: in $([ab]^i \sqcup\!\sqcup [cd]^j)$, any number of cd can occur between two occurrences of ab, but not inside one ab.

AutoIntersectMultiPair$(A_1^{(n)}, i^{(r)}, j^{(r)}, (\delta^{\mathsf{max}}_{\langle i,j\rangle})^{(r)}) \to A^{(n)}$:
1 $A^{(n)} \leftarrow \langle \Sigma \leftarrow \Sigma_1,\ Q \leftarrow \emptyset,\ \mathcal{K} \leftarrow \mathcal{K}_1,\ E^{(n)} \leftarrow \emptyset,\ \lambda,\ \rho \rangle$
2 $Stack \leftarrow \emptyset$
3 for $\forall q_1 \in Q_1 : \lambda(q_1) \neq \bar{0}$ do
4 getPushState$(q_1, (\varepsilon^{(r)}, \varepsilon^{(r)}))$
5 while $Stack \neq \emptyset$ do
6 $q \leftarrow pop(Stack)$
7 $q_1 \leftarrow \nu[q]$
8 $(s^{(r)}, u^{(r)}) \leftarrow \xi[q]$
9 for $\forall e_1 \in E(q_1)$ do
10 $(s'^{(r)}, u'^{(r)}) \leftarrow$ getLeftoverStrings$(s^{(r)} \cdot \pi_{i^{(r)}}(\ell(e_1)),\ u^{(r)} \cdot \pi_{j^{(r)}}(\ell(e_1)))$
11 if $\forall h \in [\![1, r]\!] : \big(s'_h = \varepsilon \vee u'_h = \varepsilon\big) \wedge \big(\ \big||s'_h| - |u'_h|\big| \leq (\delta^{\mathsf{max}}_{\langle i_h, j_h\rangle})_h\ \big)$
12 then $q' \leftarrow$ getPushState$(\ n(e_1), (s'^{(r)}, u'^{(r)}))$
13 $E \leftarrow E \cup \{\ \langle q, \ell(e_1), w(e_1), q' \rangle\ \}$
14 return $A^{(n)}$

getLeftoverStrings$(\dot{s}^{(r)}, \dot{u}^{(r)}) \to (s'^{(r)}, u'^{(r)})$:
15 $x^{(r)} \leftarrow longestCommonPrefix(\dot{s}^{(r)}, \dot{u}^{(r)})$
16 return $((x^{(r)})^{-1} \cdot \dot{s}^{(r)}, (x^{(r)})^{-1} \cdot \dot{u}^{(r)})$

getPushState$(q_1, (s'^{(r)}, u'^{(r)})) \to q'$:
17 if $\exists q \in Q : \nu[q] = q_1 \wedge \xi[q] = (s'^{(r)}, u'^{(r)})$
18 then $q' \leftarrow q$
19 else $q' \leftarrow createNewState(\ )$
20 $\nu[q'] \leftarrow q_1$
21 $\xi[q'] \leftarrow (s'^{(r)}, u'^{(r)})$
22 if $s'^{(r)} = \varepsilon^{(r)} \wedge u'^{(r)} = \varepsilon^{(r)}$
23 then $\lambda(q') \leftarrow \lambda(q_1)$
24 $\rho(q') \leftarrow \rho(q_1)$
25 else $\lambda(q') \leftarrow \bar{0}$
26 $\rho(q') \leftarrow \bar{0}$
27 $Q \leftarrow Q \cup \{q'\}$
28 $push(Stack, q')$
29 return q'

Then, we attempt to copy the three outgoing transitions of $q_1 = 0$ with their original labels and weights, as well as their respective target states. The $\xi[n(e)]$ of the target state of a transition e results from the $\xi[p(e)]$ of its source state, concatenated with the relevant components of its label $\ell(e)$. The longest common prefix[14] of the two string tuples in $\xi[n(e)]$ is removed. A target q that has the same $\nu[q]$ and $\xi[q]$ as an existing state q', it is not created and q' is used instead. For example, for the cyclic transition e on $q=2$ (Figure 1b), the leftover tuples of the source, $\xi[p(e)] = (\langle a, c\rangle, \langle \varepsilon, \varepsilon\rangle)$, are concatenated with the relevant

[14] The longest common prefix of two string tuples is compiled element-wise.

projections of the label, $\pi_{\langle 1,3\rangle}(\ell(e)) = \langle a, dc\rangle$ and $\pi_{\langle 2,4\rangle}(\ell(e)) = \langle a, cd\rangle$, yielding $\xi' = (\langle aa, cdc\rangle, \langle a, cd\rangle)$; since $lcp(\xi') = \langle a, cd\rangle$, the leftover tuples of the target are finally $\xi[n(e)] = (\langle a, c\rangle, \langle \varepsilon, \varepsilon\rangle)$, which implies that $p(e) = n(e)$.

State $q = 3$ (resp. $q = 1$) and its incoming transition are not created because $\delta^{\mathsf{max}}_{\langle 1,2\rangle}$ is exceeded (resp. dc and cd are incompatible leftover strings). State $q = 9$ is non-final, although $\nu[9] = 2$ is final, because its leftover tuples are not $(\langle \varepsilon, \varepsilon\rangle, \langle \varepsilon, \varepsilon\rangle)$. As expected, the support $a{:}\varepsilon{:}c{:}\varepsilon\ (a{:}a{:}dc{:}cd)^*\ ba{:}ab{:}c{:}\varepsilon\ \ \varepsilon{:}a{:}\varepsilon{:}cc$ of the auto-intersection is equal to the set $\{\ \langle a^{i+1}ba, a^{i+1}ba, (cd)^i c^2, (cd)^i c^2\rangle \mid i \in \mathbb{N}\ \}$.

Algorithm: The Algorithm AutoIntersectMultiPair computes the auto-intersection $\sigma_{\{i_1=j_1,\dots i_r=j_r\}}$ in the case where $\forall h \in [\![1, r]\!]$, $\langle A_1^{(n)}, i_h, j_h\rangle \in \Theta$. The tape indices are specified in two tuples, $i^{(r)} = \langle i_1, \dots i_r\rangle$ and $j^{(r)} = \langle j_1, \dots j_r\rangle$, that are also used for projection, $\pi_{i^{(r)}} = \pi_{\langle i_1,\dots i_r\rangle}$. The delay limits, related to the two index tuples, are specified in one tuple, $(\delta^{\mathsf{max}}_{\langle i,j\rangle})^{(r)} = \langle(\delta^{\mathsf{max}}_{\langle i_1,j_1\rangle})_1, \dots (\delta^{\mathsf{max}}_{\langle i_r,j_r\rangle})_r\rangle$. The function GetPushState checks whether a target state already exists or not; a new state is created if necessary and pushed onto the stack.

The construction of $A^{(n)} = \sigma_{\{i_1=j_1,\dots i_r=j_r\}}(A_1^{(n)})$ is guaranteed to terminate because each auto-intersection $\sigma_{\{i_h=j_h\}}$ terminates. Only such states are created for $\sigma_{\{i_1=j_1,\dots i_r=j_r\}}$, that would also have been created for each $\sigma_{\{i_h=j_h\}}$ separately. Therefore, the number $|Q|$ of states in $A^{(n)}$ cannot exceed that of each separate auto-intersection. Finally we get $|Q| \ < \ 2\,|Q_1|\,\frac{|\Sigma_1|^{\min_h(\delta^{\mathsf{max}}_{\langle i_h,j_h\rangle})}-1}{|\Sigma_1|-1}$.

4.4 Multi-pair auto-intersection: iterative construction

We now address the case of a multi-pair auto-intersection $\sigma_{\{i_1=j_1,\dots i_r=j_r\}}$ such that there may exist $h \in [\![1, r]\!]$ with $\langle A_1^{(n)}, i_h, j_h\rangle \notin \Theta$. As an example we consider the WFSM $A_1^{(4)}$ of Figure 2a and the auto-intersection $\sigma_{\{1=2,3=4\}}(A_1^{(4)})$. The support $(a{:}a{:}dc{:}cd\ \cup\ a{:}\varepsilon{:}c{:}\varepsilon)^*\ (ba{:}ab{:}\varepsilon{:}c)^*\ \varepsilon{:}a{:}\varepsilon{:}c$ of $A_1^{(4)}$ is equal to the set $\{\ \langle a^{i+j}(ba)^h, a^i(ab)^h a, ([dc]^i \sqcup\!\sqcup c^j), (cd)^i c^h c\rangle \mid i, j, h \in \mathbb{N}\ \}$.
Since $\langle A_1^{(4)}, 1, 2\rangle \in \Theta$ with $\delta^{\mathsf{max}}_{\langle 1,2\rangle} = 1$ and $\langle A_1^{(4)}, 3, 4\rangle \notin \Theta$, $\sigma_{\{1=2\}}(A_1^{(4)})$ is first compiled (Figure 2b); its support $(a{:}a{:}dc{:}cd)^*a{:}\varepsilon{:}c{:}\varepsilon\ (a{:}a{:}dc{:}cd)^*(ba{:}ab{:}\varepsilon{:}c)^*\varepsilon{:}a{:}\varepsilon{:}c$ is the set $\{\ \langle a^{i+j+1}(ba)^h, a^{i+j+1}(ba)^h, (dc)^i(cd)^j c, (cd)^i(cd)^j c^{h+1}\rangle \mid i, j, h \in \mathbb{N}\ \}$.
Since $\langle \sigma_{\{1=2\}}(A_1^{(4)}), 3, 4\rangle \in \Theta$ with $\delta^{\mathsf{max}}_{\langle 3,4\rangle} = 2$, we now can compile the second auto-intersection (Figure 2c), whose support $a{:}\varepsilon{:}c{:}\varepsilon\ (a{:}a{:}dc{:}cd)^*\ \varepsilon{:}a{:}\varepsilon{:}c$ is equal to the set $\{\ \langle a^{i+1}, a^{i+1}, (cd)^i c, (cd)^i c\rangle \mid i \in \mathbb{N}\ \}$.

Algorithm: The Algorithm AutoIntersection attempts to construct iteratively the auto-intersection $\sigma_T(A_1^{(n)})$ on tape pairs specified by the set T. The function FilterTapePairs implements the filtering of $\sigma_T(A_1^{(n)})$ and the function SelectTapePairs selects tapes satisfying $\langle A_1^{(n)}, i, j\rangle \in \Theta$. The function CompileDelayLimit computes the limit $\delta^{\mathsf{max}}_{\langle i,j\rangle}$.

As long as T is not empty (Line 2), the algorithm filters all tape pairs (see Section 4.2) then selects all constraints $t = (i = j)$ on which the auto-intersection

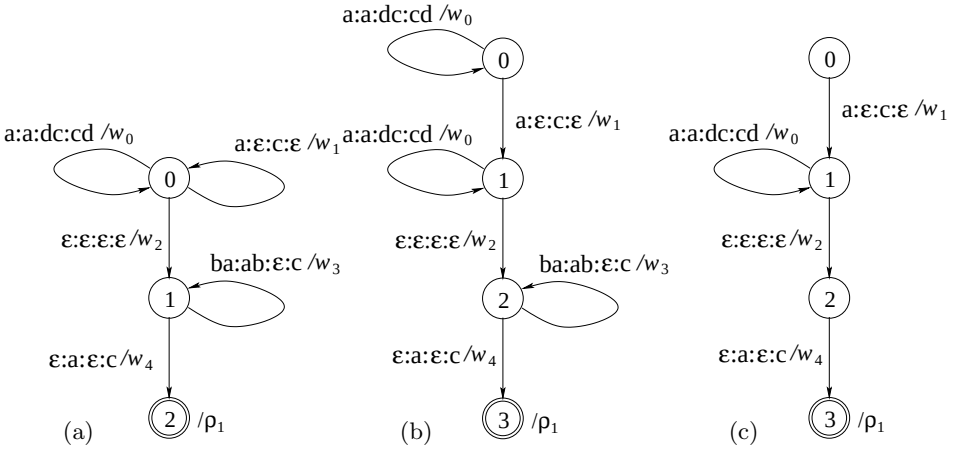

Fig. 2. Iterative compilation of auto-intersection: (a) a WFSM $A_1^{(4)}$, (b) its auto-intersection $\sigma_{\{1=2\}}(A_1^{(4)})$, and (c) a second auto-intersection $\sigma_{\{3=4\}}(\sigma_{\{1=2\}}(A_1^{(4)}))$.

is constructible (Line 4), and compiles a limit of the delay $\delta^{\mathsf{max}}_{\langle i,j\rangle}$ for each of those pairs (Line 8–9). Finally, it constructs an auto-intersection simultaneously on all selected pairs (Line 10–13). In the next iteration, it tries the same for the set of remaining pairs (Line 14, 2). The test of constructibility may now succeed on a pair of tapes on which it previously failed, because the cycles that made it fail may have disappeared in between. The algorithm terminates either successfully if all tape pairs can been processed ($T = \emptyset$) or not if some pairs remain ($T \neq \emptyset \wedge T' = \emptyset$). In the latter case, an error code is returned (Line 5–6).

AutoIntersection$(A_1^{(n)}, T) \to A^{(n)}$: [$T = \{t=(i=j)\}$; $A^{(n)} = \sigma_T(A_1^{(n)})$]

1 $A^{(n)} \leftarrow A_1^{(n)}$
2 while $T \neq \emptyset$ do
3 $A^{(n)} \leftarrow$ filterTapePairs$(A^{(n)}, T)$
4 $T' \leftarrow$ selectTapePairs$(A^{(n)}, T)$
5 if $T' = \emptyset$
6 then return $\perp$ [error code]
7 else $i^{(r=0)} \leftarrow j^{(r=0)} \leftarrow (\delta^{\mathsf{max}}_{\langle i,j\rangle})^{(r=0)} \leftarrow \langle\,\rangle$
8 for $\forall t=(i=j) \in T'$ do
9 $\delta^{\mathsf{max}}_{\langle i,j\rangle} \leftarrow$ compileDelayLimit$(A^{(n)}, i, j)$
10 $i^{(r+1)} \leftarrow$ *append*$(\, i^{(r)}, i \,)$
11 $j^{(r+1)} \leftarrow$ *append*$(\, j^{(r)}, j \,)$
12 $(\delta^{\mathsf{max}}_{\langle i,j\rangle})^{(r+1)} \leftarrow$ *append*$(\, (\delta^{\mathsf{max}}_{\langle i,j\rangle})^{(r)}, \delta^{\mathsf{max}}_{\langle i,j\rangle} \,)$
13 $A^{(n)} \leftarrow$ AutoIntersectMultiPair$(A^{(n)}, i^{(r)}, j^{(r)}, (\delta^{\mathsf{max}}_{\langle i,j\rangle})^{(r)})$
14 $T \leftarrow T \setminus T'$
15 return $A^{(n)}$

---

5 Conclusion

We conclude by briefly describing an improved version of the Algorithm JOIN1. It is based on the operation of single-pair equi-join.[15] A single-pair join $A_1^{(n)} \bowtie_{\{i=j\}} A_2^{(m)}$ can be compiled in one step, rather than first building the cross-product, $A_1^{(n)} \times A_2^{(m)}$, and then deleting most of its paths by the auto-intersection $\sigma_{\{i=n+j\}}$. Our single-pair join algorithm is very similar to the classical transducer composition; it simulates the behaviour of an ε-filter (cf [15]) for aligning ε-transitions in the two transducers.

The improved join algorithm selects arbitrarily one pair of tapes and performs on it a single-pair equi-join (that always yields a rational result, at least for weights over a commutative semiring) followed by an auto-intersection for the remaining pairs (that may fail). So far we found no evidence that would allow us to decide whether the choice of the first pair of tapes, that is used in the equi-join, matters for the success of the whole algorithm.

Acknowledgments

We wish to thank Jason Eisner for allowing us to use a bulk of relevant notation that he elaborated (cf. Footnote 4), Mark-Jan Nederhof for pointing out the relationship between auto-intersection and Post's Correspondence Problem (personal communication), and the anonymous reviewers of our paper for their valuable advice.

References

1. Samuel Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, San Diego, 1974.
2. Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proc. of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, 2002.
3. Calvin C. Elgot and Jorge E. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965.
4. Christiane Frougny and Jacques Sakarovitch. Synchronized rational relations of finite and infinite words. *Theoretical Computer Science*, 108(1):45–82, 1993.
5. Tero Harju and Juhani Karhumäki. The equivalence problem of multitape finite automata. *Theoretical Computer Science*, 78(2):347–355, 1991.
6. Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.
7. Martin Kay. Nonconcatenative finite-state morphology. In *Proc. 3rd Int. Conf. EACL*, pages 2–10, Copenhagen, Denmark, 1987.
8. André Kempe. NLP applications based on weighted multi-tape automata. In *Proc. 11th Conf. TALN*, pages 253–258, Fes, Morocco, 2004.

[15] According to database notation an equi-join does not discard any tape.

9. André Kempe, Jean-Marc Champarnaud, and Jason Eisner. A note on join and auto-intersection of n-ary rational relations. In B. Watson and L. Cleophas, editors, *Proc. Eindhoven FASTAR Days*, number 04–40 in TU/e CS TR, pages 64–78, Eindhoven, Netherlands, 2004.
10. André Kempe, Jean-Marc Champarnaud, Jason Eisner, Franck Guingne, and Florent Nicart. A class of rational n-WFSM auto-intersections. In O. H. Ibarra and Z. Dang, editors, *Proc. 10th Int. Conf. CIAA*, Sophia Antipolis, France, 2005. *(to appear)*.
11. André Kempe, Franck Guingne, and Florent Nicart. Algorithms for weighted multi-tape automata. Research report 2004/031, Xerox Research Centre Europe, Meylan, France, 2004.
12. George Anton Kiraz. Multitiered nonlinear morphology using multitape finite automata: a case study on Syriac and Arabic. *Computational Lingistics*, 26(1):77–105, 2000.
13. Werner Kuich and Arto Salomaa. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer Verlag, Berlin, Germany, 1986.
14. Mehryar Mohri. 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, 2003. Springer Verlag, Berlin, Germany.
15. Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. A rational design for a weighted finite-state transducer library. *Lecture Notes in Computer Science*, 1436:144–158, 1998.
16. Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.
17. Arnold L. Rosenberg. On n-tape finite state acceptors. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 76–81, 1964.