A Class of Rational n-WFSM Auto-Intersections

André Kempe[1], Jean-Marc Champarnaud[2], Jason Eisner[3], Franck Guingne[1,4], and Florent Nicart[1,4]

[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] Johns Hopkins University – Computer Science Department
3400 N. Charles St. – Baltimore, MD 21218 – United States
jason@cs.jhu.edu – http://www.cs.jhu.edu/∼jason/

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

Abstract. Weighted finite-state machines with n tapes describe n-ary rational string relations. The join n-ary relation is very important in applications. It is shown how to compute it via a more simple operation, the auto-intersection. Join and auto-intersection generally do not preserve rationality. We define a class of triples $\langle A, i, j\rangle$ such that the auto-intersection of the machine A on tapes i and j can be computed by a delay-based algorithm. We point out how to extend this class and hope that it is sufficient for many practical applications.

1 Introduction

Multi-tape finite-state machines (FSMs) [1–5] are a natural generalization of the familiar finite-state acceptors (one tape) and transducers (two tapes). Multi-tape machines have been used in the morphology of Semitic languages, to synchronize the vowels, consonants, and templatic pattern into a surface form [3, 6].

The n-ary relation defined by a (weighted) n-tape FSM is a (weighted) *rational* relation. *Finite* relations are defined by *acyclic* FSMs, and are well-studied since they can be viewed as relational databases whose fields are strings [7]. E.g., a two-column database can be represented by an acyclic finite-state transducer.

Unfortunately, one pays a price for generalizing to multi-column databases with *infinitely* many rows, as defined by *cyclic* FSMs. Cyclic FSMs are closed under the rational operations, but not under all relational operations, as finite databases are. For example, transducers are not closed under intersection [1].

In this paper we consider a practically useful generalization of transducer intersection, *multi-tape join*, which is analogous to *natural join* of databases. More precisely, we study an equivalent but simpler problem, *auto-intersection*. The emptiness or rationality of the result is generally undecidable [7]. Therefore

we define a simple class Θ of triples $\langle A, i, j\rangle$ such that the auto-intersection of the machine A w.r.t. tapes i and j is rational. Our auto-intersection algorithm for this class is based on the notion of delay [8, 9]. We focus on the case of an auto-intersection w.r.t. two tapes, which is sufficient to explain the basic ideas and problems, and we briefly discuss the general case. We conclude by pointing out possible extensions of the class Θ.

Weighted n-ary relations and their machines are introduced in Section 2. Join and auto-intersection operations are presented in Section 3. A class of compilable auto-intersections and the associated algorithm are defined in Section 4.

2 Definitions

We recall some definitions about n-ary weighted relations and their machines, following the usual definitions for multi-tape automata [2, 10], with semiring weights added just as for acceptors and transducers [11, 12]. See [7] for details.

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 $\mathcal{K} = \langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1}\rangle$. A relation assigns a weight to any n-tuple of strings. A weight of $\bar{0}$ can be interpreted as meaning that the tuple is not in the relation.[1] 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, which we now define.

We adopt the convention that variable names referring to n-tuples of strings include a superscript $^{(n)}$. Thus we write $s^{(n)}$ rather than $\vec{s}$ for a tuple of strings $\langle s_1, \ldots s_n\rangle$. We also use this convention for the names of objects that contain n-tuples of strings, such as n-tape machines and their transitions and paths.

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. We say that $q \in Q$ is an initial state if $\lambda(q) \neq \bar{0}$, and a final state if $\varrho(q) \neq \bar{0}$.

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 $e_1^{(n)} e_2^{(n)} \cdots e_k^{(n)}$ where $n(e_i^{(n)}) = p(e_{i+1}^{(n)})$ for all $i \in [\![1, k-1]\!]$. The path's label $\ell(\gamma^{(n)})$ is the element-wise

[1] 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.

concatenation of the labels of its transitions. The path's weight $w(\gamma^{(n)})$ is

$$w(\gamma^{(n)}) \stackrel{\text{def}}{=} \lambda(p(e_1^{(n)})) \otimes \left(\bigotimes_{j \in [\![1,k]\!]} w\left(e_j^{(n)}\right) \right) \otimes \varrho(n(e_k^{(n)})) \tag{1}$$

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 (if any) that accept the n-tuple of strings $s^{(n)}$. Now, 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:

$$\mathcal{R}_{A^{(n)}}(s^{(n)}) \stackrel{\text{def}}{=} \bigoplus_{\gamma^{(n)} \in \Gamma_{A^{(n)}}(s^{(n)})} w(\gamma^{(n)}) \tag{2}$$

3 Operations

We now describe some central operations on n-ary weighted relations and their n-WFSMs [13]. The auto-intersection operation is introduced, with the aim of simplifying the computation of the join operation. Our notation is inspired by relational databases. For mathematical details of simple operations see [7].

Simple Operations: The set of n-ary weighted rational relations can be constructed as the closure of the elementary n-ary weighted relations (those whose support consists of at most one tuple) under the basic rational operations of *union*, *concatenation* and *Kleene closure*. These rational operations can be implemented by simple constructions on the corresponding nondeterministic n-tape WFSMs [14]. 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. Since tapes must explicitly be selected by number, join is neither associative nor commutative.

For any distinct $i_1,\dots i_r \in [\![1,n]\!]$ and any distinct $j_1,\dots j_r \in [\![1,m]\!]$, we define a *join* operator $\bowtie_{\{i_1=j_1,\dots i_r=j_r\}}$. It combines an n-ary and an m-ary relation into an $(n+m-r)$-ary relation defined as follows:[2]

$$\left(\mathcal{R}_1^{(n)} \bowtie_{\{i_1=j_1,\dots i_r=j_r\}} \mathcal{R}_2^{(m)}\right)(\langle u_1,\dots u_n, s_1,\dots s_{m-r}\rangle) \stackrel{\text{def}}{=} \mathcal{R}_1^{(n)}(u^{(n)}) \otimes \mathcal{R}_2^{(m)}(v^{(m)}) \tag{3}$$

[2] 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.

$v^{(m)}$ 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 $\bowtie_{\{1=1,2=2,\dots n=n\}}$. Examples of *single-tape join* (where $r = 1$) 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. The cross product $\times$, as in $\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)}$, can be expressed as $\bowtie_{\emptyset}$, the join of no tapes ($r = 0$). Our main concern in this paper is multi-tape join ($r > 1$).

Some practical applications require the multi-tape join operation, for example: probabilistic normalization of n-WFSMs conditioned on r tapes,[3] or searching for cognates [16]. Unfortunately, rational relations are *not* closed under arbitrary joins [7]. The join operation is so useful that it is helpful to have a partial algorithm: hence our motivation for studying auto-intersection.

Auto-Intersection: For any distinct $i_1, j_1, \dots\ i_r, j_r \in [\![1,n]\!]$, we define an *auto-intersection* operator $\sigma_{\{i_1=j_1,i_2=j_2,\dots i_r=j_r\}}$. 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.[4] The formal definition is:

$$\left(\sigma_{\{i_1=j_1,\dots i_r=j_r\}}(\mathcal{R}^{(n)})\right)(\langle s_1,\dots s_n\rangle) \stackrel{\text{def}}{=} \begin{cases} \mathcal{R}^{(n)}(\langle s_1,\dots s_n\rangle) & \text{if } (\forall k \in [\![1,r]\!]) s_{i_k} = s_{j_k} \\ \bar{0} & \text{otherwise} \end{cases} \tag{4}$$

It is easy to check that auto-intersecting a relation is different from joining the relation with its own projections. Actually, join and auto-intersection are related by the following equalities:

$$\mathcal{R}_1^{(n)} \bowtie_{\{i_1=j_1,\dots i_r=j_r\}} \mathcal{R}_2^{(m)} \ = \ \overline{\pi}_{\{n+j_1,\dots n+j_r\}} \left(\sigma_{\{i_1=n+j_1,\dots i_r=n+j_r\}}(\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)}) \right) \tag{5}$$

$$\sigma_{\{i_1=j_1,\dots i_r=j_r\}}(\mathcal{R}^{(n)}) \ = \ \mathcal{R}^{(n)} \bowtie_{\{i_1=1,j_1=2,\dots i_r=2r-1,j_r=2r\}} \left(\underbrace{(\pi_{\langle 1,1\rangle}(\Sigma^*) \times \dots \times \pi_{\langle 1,1\rangle}(\Sigma^*)}_{r \text{ times}} \right) \tag{6}$$

Thus, for any class of difficult join instances whose results are non-rational or have undecidable properties [7], 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.

The case $r = 1$ is *single-pair* auto-intersection. An auto-intersection on multiple pairs of tapes ($r > 1$) can be defined in terms of multiple single-pair auto-intersections:

$$\sigma_{\{i_1=j_1,\dots i_r=j_r\}}(\mathcal{R}^{(n)}) \ \stackrel{\text{def}}{=} \ \sigma_{\{i_r=j_r\}}(\cdots \sigma_{\{i_1=j_1\}}(\mathcal{R}^{(n)}) \cdots) \tag{7}$$

[3] This is a straightforward generalization of J. Eisner's construction for probabilistic normalization of transducers ($n = 2$) conditioned on one tape ($r = 1$) [15].

[4] The requirement that the $2r$ indices be distinct mirrors the similar requirement on join and is needed for (6) to hold. But it can be evaded by duplicating tapes: the illegal operation $\sigma_{\{1=2,2=3\}}(\mathcal{R})$ can be computed as $\overline{\pi}_{\{3\}}(\sigma_{\{1=2,3=4\}}(\pi_{\langle 1,2,2,3\rangle}(\mathcal{R})))$.

Nonetheless, 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. The reason is that even when $\sigma_{\{i_1=j_1,\ldots i_r=j_r\}}$ is rational, a finite-state strategy for computing it via (7) could fail by encountering non-rational intermediate results. For example, consider applying $\sigma_{\{2=3,4=5\}}$ to the rational 5-ary relation $\{\langle a^i b^j, c^i, c^j, x, y\rangle \mid i, j \in \mathbb{N}\}$. The final result is rational (the empty relation), but the intermediate result after applying just $\sigma_{\{2=3\}}$ would be the non-rational relation $\{\langle a^i b^i, c^i, c^i, x, y\rangle \mid i \in \mathbb{N}\}$.

4 Single-pair auto-intersection

As indicated by (5), a join can be computed via an auto-intersection, which can be decomposed as a sequence of single-pair auto-intersections as in (7). We therefore focus on the single-pair case, which is sufficient to explain the basic ideas and problems. As a consequence of Post's Correspondence Problem, there exists no fully general algorithm for auto-intersection [7]. We show that it is however possible to compile the auto-intersection $\sigma_{\{i=j\}}(A)$ for a limited class of triples $\langle A, i, j\rangle$ whose definition is based on the notion of delay.

By *delay* we mean the difference of length of two strings of an n-tuple:[5] $\delta_{\langle i,j\rangle}(s^{(n)}) = |s_i| - |s_j|$ (with $i, j \in [\![1, n]\!]$). The delay of a path γ is determined from its respective labels on tapes i and j: $\delta_{\langle i,j\rangle}(\gamma) = |\ell_i(\gamma)| - |\ell_j(\gamma)|$.

For any $\mathcal{R}_1^{(n)}$, its autointersection $\mathcal{R}^{(n)} = \sigma_{\{i=j\}}(\mathcal{R}_1^{(n)})$ assigns a weight $\bar{0}$ to each string tuple $s^{(n)}$ such that $s_i \neq s_j$. For simplicity, our auto-intersection construction will ensure this by never creating any successful paths γ for which $\ell_i(\gamma) \neq \ell_j(\gamma)$. One consequence is that all successful paths of our constructed $A^{(n)} = \sigma_{\{i=j\}}(A_1^{(n)})$, where $A_1^{(n)}$ expresses $\mathcal{R}_1^{(n)}$, will have a delay equal to 0 : $\forall \gamma \in \Gamma_{A^{(n)}},\ \ell_i(\gamma) = \ell_j(\gamma) \Rightarrow |\ell_i(\gamma)| = |\ell_j(\gamma)| \ \Rightarrow\ \delta_{\langle i,j\rangle}(\gamma) = 0$.

To be more specific, let $\Gamma^0 \subseteq \Gamma_{A_1^{(n)}}$ be the set of successful paths of $A_1^{(n)}$ with a delay of 0. Then our construction will "copy" an appropriate subset of Γ^0 into the constructed $A^{(n)}$. Note that $\forall \gamma = \gamma_1 \gamma_2 \cdots \gamma_r \in \Gamma^0,\ \sum_{h=1}^{r} \delta_{\langle i,j\rangle}(\gamma_h) = \delta_{\langle i,j\rangle}(\gamma) = 0$.

4.1 Bounded delay auto-intersection

We now focus temporarily on n-WFSMs such as $A_1^{(n)}$ in Figure 1, whose cycles all have a positive delay with respect to the tapes i, j of the single-pair auto-intersection.

Such an n-WFSM might contain paths with arbitrarily large delay. However, if we consider only its paths $\gamma \in \Gamma^0$, it turns out that they must have *bounded delay*. That is, that there is a bound $\delta^{\mathsf{max}}_{\langle i,j\rangle}(A_1^{(n)})$ for the WFSM such that $|\delta_{\langle i,j\rangle}(\gamma_1)| \leq \delta^{\mathsf{max}}_{\langle i,j\rangle}(A_1^{(n)})$ for any prefix γ_1 of any $\gamma \in \Gamma^0$.

In this section, we outline how to compute the bound $\delta^{\mathsf{max}}_{\langle i,j\rangle}(A_1^{(n)})$. Then, while the algorithm of the next section (4.2) is copying paths from $A_1^{(n)}$, it can avoid

[5] We use the notion of delay similarly as in the synchronization of transducers [8, 9].

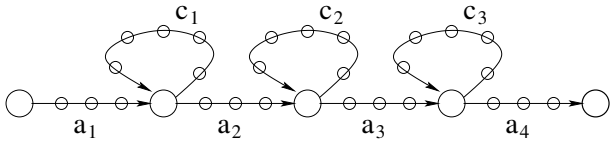

Fig. 1. An example n-WFSM $A_1^{(n)}$, having four acyclic factors a_h and three cycles c_h with positive delay.

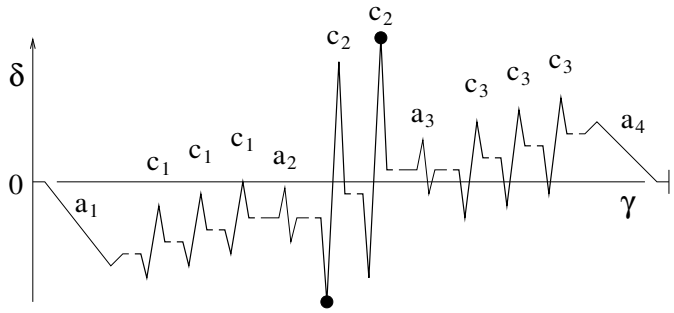

Fig. 2. Hypothetical monitoring of the delay of successively longer prefixes γ_1 of one path γ through $A_1^{(n)}$ whose total delay $\delta_{\langle i,j\rangle}(\gamma) = 0$. Global extrema are marked. By assumption, each of the cycles c_1, c_2, c_3 has positive delay.

extending any prefix whose delay's absolute value exceeds $\delta^{\mathsf{max}}_{\langle i,j\rangle}(A_1^{(n)})$. (Such a prefix is useless because it will not extend into a path in Γ^0, let alone a path with $\ell_i(\gamma) = \ell_j(\gamma)$.)

If we plotted the delay for successively longer prefixes γ_1 of a given path $\gamma \in \Gamma^0$, as γ_1 ranges from ϵ to γ, we would obtain a curve that begins and ends with delay $\delta_{\langle i,j\rangle}(\gamma_1) = 0$, as shown in Figure 2. How can we bound the maximum $\hat{\delta}_{\langle i,j\rangle}(\gamma_1)$ and minimum $\check{\delta}_{\langle i,j\rangle}(\gamma_1)$ along this curve?

A lower bound is given by $\check{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)}) \leq 0$, defined as the minimum delay of any *acyclic* path that begins at an initial state of $A_1^{(n)}$. Why? Since $\gamma \in \Gamma^0$ is a successful path, any prefix γ_1 of γ can be regarded as an acyclic path of this sort with zero or more cycles inserted. But these cycles can only increase the total delay (by the assumption that their delay is positive), so $\delta_{\langle i,j\rangle}(\gamma_1) \geq \check{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})$.

An upper bound is given by $\hat{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)}) \geq 0$, defined as the *negation* of the minimum delay of any acyclic path that ends at a final state of $A_1^{(n)}$. By symmetry, that minimum delay is a lower bound on the delay of any *suffix* γ_2 of γ. But if we factor $\gamma = \gamma_1\gamma_2$, we have $\delta_{\langle i,j\rangle}(\gamma_1) + \delta_{\langle i,j\rangle}(\gamma_2) = \delta_{\langle i,j\rangle}(\gamma) = 0$, since $\gamma \in \Gamma^0$. It follows that $\hat{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})$ is an upper bound on the delay of γ_1.

The minimum $\check{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})$ is finite because there are only finitely many acyclic paths from initial states to consider. $\hat{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})$ is similar. Exhaustively considering all these acyclic paths by backtracking, as illustrated in Figure 3, takes

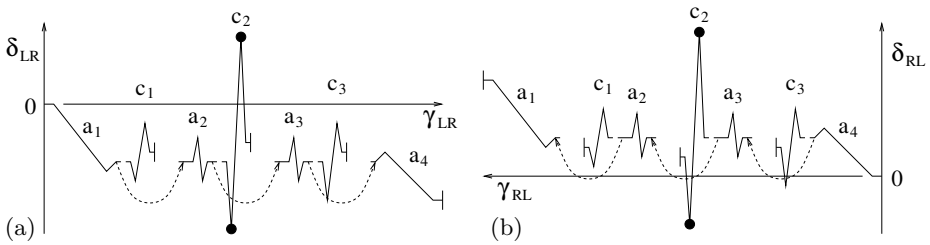

Fig. 3. Monitoring the delay on all acyclic paths of $A_1^{(n)}$, exploring (a) forward from initial states and (b) backward from final states. In (b), the sign of the delay is negated. Global extrema are marked. Gaps denote points where the search algorithm backtracked to avoid completing a cycle. Dashed arrows lead from a choice point to alternative paths that are explored after backtracking.

exponential time in the worst case.[6] However, that is presumably unavoidable since the decision problem associated with finding $\check{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})$ is NP-complete (by a trivial reduction from Hamiltonian Path).

Visually, *all* acyclic prefix paths are represented in Figure 3a, so a given acyclic prefix path must fall entirely above the minimum of Figure 3a. A possibly cyclic prefix path as in Figure 2 can only be higher still, since all cycles have positive delay. A visual argument can also be made from Figure 3b.

These prefix-delay bounds, $\delta_{\langle i,j\rangle}(\gamma_1) \in [\![\check{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)}), \hat{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})]\!]$, in fact apply whenever γ_1 is a prefix of a $\gamma \in I^0$ that traverses no cycle of negative delay. If on the other hand γ traverses no cycle of *positive* delay, we have similarly $\delta_{\langle i,j\rangle}(\gamma_1) \in [\![\check{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)}), \hat{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})]\!]$, where these bounds are found by considering maximum rather than minimum delays. In either case, we see that

$$|\delta_{\langle i,j\rangle}(\gamma_1)| \le \delta^{\mathsf{max}}_{\langle i,j\rangle}(A_1^{(n)}) \tag{8}$$

$$\stackrel{\text{def}}{=} \max\left(|\hat{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})|\,,\ |\hat{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})|\,,\ |\check{\delta}^{LR}_{\langle i,j\rangle}(A_1^{(n)})|\,,\ |\check{\delta}^{RL}_{\langle i,j\rangle}(A_1^{(n)})|\right) \tag{9}$$

Definition of the class: 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 (with respect to tapes i and j). The Algorithm AutoIntersectSinglePair (see Section 4.2) computes the auto-intersection $A^{(n)} = \sigma_{\{i=j\}}(A_1^{(n)})$ for any triple in Θ, thanks to the property that it has a delay not exceeding the limit $\delta^{\mathsf{max}}_{\langle i,j\rangle}(A_1^{(n)})$ defined in (9).

[6] In practice, one would first trim $A_1^{(n)}$ to remove edges and states that do not appear on any successful path. This may reduce the problem size, without affecting the defined relation or its auto-intersection.

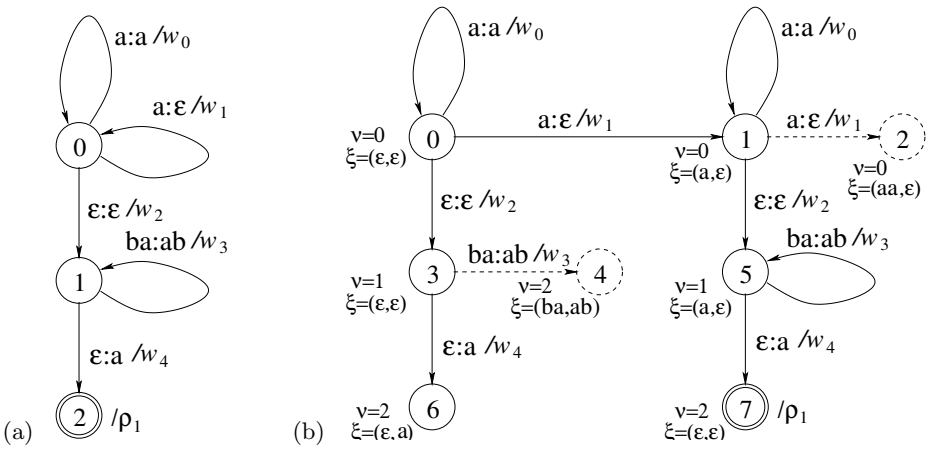

Fig. 4. (a) An n-WFSM $A_1^{(2)}$ and (b) its auto-intersection $A^{(2)} = \sigma_{\{1=2\}}(A_1^{(2)})$ (dashed parts are not constructed).

4.2 Algorithm for bounded delay auto-intersection

We take first the example of the n-WFSM $A_1^{(2)}$ of Figure 4a. The triple $\langle A_1^{(2)}, 1, 2\rangle$ is obviously in the class Θ. The delay of the auto-intersection $A^{(2)} = \sigma_{\{1=2\}}(A_1^{(2)})$ is bounded by $\delta^{\max}_{\langle 1,2\rangle}(A_1^{(2)}) = 1$. The support $((a{:}a \cup a{:}\varepsilon)^* (ba{:}ab)^* \varepsilon{:}a)$ of $A_1^{(2)}$ is equal to $\{ \langle a^{i+j}(ba)^h, a^i(ab)^h a\rangle \mid i,j,h \in \mathbb{N} \}$.

To construct the auto-intersection,[7] we copy states and transitions one by one from $A_1^{(2)}$ (Figure 4a) to $A^{(2)}$ (Figure 4b), starting with the initial states. We assign to each state q of $A^{(2)}$ two variables: $\nu[q] = q_1$ is the associated state of $A_1^{(2)}$, and $\xi[q] = (s,u)$ gives the "leftover strings" of the path read while reaching q: s has been read on tape i but not yet on tape j, and vice-versa for u. (Thus the delay accumulated so far is $|s| - |u|$. In practice either s or u will be ϵ.)

In our example, we start at the initial state $q_1 = 0$, with $\nu[0] = 0$ and $\xi[0] = (\varepsilon, \varepsilon)$. Then, we copy the three outgoing transitions of $q_1 = 0$, with their original labels and weights, as well as creating their respective target states with appropriate ν and ξ. If a target state has already been created with this ν and ξ, we reuse it. If not, we create it and proceed to copy *its* outgoing transitions.

The target state of a transition e has an $\xi[n(e)]$ that is obtained from the $\xi[p(e)]$ of its source state, concatenated with the relevant components of its label

[7] Our construction bears resemblance to known transducer synchronization procedures. The algorithm of Frougny and Sakarovitch [8] and Mohri's algorithm [9] can, however, not cope with n-FSMs having unbounded delay, such as the one in Figure 4a. Furthermore, they generate synchronized n-FSMs, which is not necessarily what one is aiming for. The algorithm [8] is based on a $\mathbb{K}$-covering of the transducer. Our algorithm is based on a general reachability-driven construction, as [9], 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.

$\ell(e)$. The longest common prefix of s and u in $\xi[n(e)] = (s, u)$ is then removed. For example, for the cyclic transition e on $q = 5$ (a copy of that on $q_1 = 1$), the leftover strings of the target are $\xi[n(e)] = \langle ab, ab\rangle^{-1}(\langle a, \varepsilon\rangle\langle ba, ab\rangle) = \langle a, \varepsilon\rangle$. Also, $\nu[n(e)] = 1$. This implies that $n(e) = p(e)$ because they have the same ξ and ν.

In Figure 4b, new state $q = 2$ and its incoming transition are not created because here the delay of 2 (determined from $\xi[q]$) has an absolute value that exceeds $\delta^{\mathsf{max}}_{\langle 1,2\rangle}(A^{(2)}_1) = 1$, which means that any path to new state $q = 2$ cannot be in $A^{(2)}$. State $q = 4$ and its incoming transitions are not created either, because both leftover strings in $\xi[4]$ are non-empty, which means that any path traversing $q = 4$ has different strings on tape 1 and 2 and can therefore not be in $A^{(2)}$. State $q = 6$ is non-final, although $q_1 = 2 = \nu[6]$ is final, because $\xi[6]$ is not $(\varepsilon, \varepsilon)$, which means that any path ending in $q = 6$ has different strings on tape 1 and 2. As expected, the support $((a{:}a)^*\ a{:}\varepsilon\ (a{:}a)^*\ (ba{:}ab)^*\ \varepsilon{:}a)$ of the constructed auto-intersection $A^{(2)}$ is equal to $\{\ \langle a^{i+j+1}(ba)^h, a^{i+j+1}(ba)^h\rangle \mid i, j, h \in \mathbb{N}\ \}$.

Algorithm: The formal algorithm AUTOINTERSECTSINGLEPAIR in Figure 5 finds the auto-intersection, provided only that $\delta^{\mathsf{max}}_{\langle i,j\rangle}(A^{(n)}_1)$ is indeed an upper bound on the absolute value of the delay of any prefix γ_1 of any successful path γ in $A^{(n)}_1$ such that $\ell_i(\gamma) = \ell_j(\gamma)$.

We have seen how to find such a bound when $\langle A^{(n)}_1, i, j\rangle$ is in the class Θ. Such a bound may also exist in other cases. Even when such a bound is not known or does not exist, one could impose one arbitrarily, in order to obtain an approximate auto-intersection.

The loop at line 5 must terminate, since a finite state set Q will be constructed for $A^{(n)}$ and each state is pushed only once. Q is finite because distinct states $q \in Q$ must have distinct values for $\nu[q]$ and/or $\xi[q]$. The number of values of $\nu[q] = q_1$ is limited by $|Q_1|$ (the number of states of A_1), and the number of values of $\xi[q] = (s, u)$ both by $|\Sigma_1|$ and $\delta^{\mathsf{max}}_{\langle i,j\rangle}$ because either s or u is empty and the other string is not longer than $\delta^{\mathsf{max}}_{\langle i,j\rangle}$. As a result, $|Q| \ < \ 2\,|Q_1|\,\frac{|\Sigma_1|^{\delta^{\mathsf{max}}_{\langle i,j\rangle}+1}-1}{|\Sigma_1|-1}$.

5 Conclusion

We conclude with two enhancements of the auto-intersection construction. Both attempt to remove cycles of A that prevent $\langle A, i, j\rangle$ from falling in Θ.

First, one can eliminate paths γ such that $\ell_i(\gamma)$ not only differs from $\ell_j(\gamma)$, but differs from $\ell_j(\gamma')$ for *all* γ' such that $A(\gamma') \neq \bar{0}$, or vice-versa. Given $\langle A^{(n)}, i, j\rangle$, define $A^{(1)}_i$ to be the projection $\pi_{\langle i\rangle}(A^{(n)})$.[8] Define $A^{(1)}_j$ similarly,

[8] More precisely, $A^{(1)}_i$ should define a "neutrally weighted" version of the projected language, in which non-$\bar{0}$ string weights have been changed to $\bar{1}$. To obtain this, replace all $\bar{0}$ and non-$\bar{0}$ weights in the weighted acceptor $\pi_{\langle i\rangle}(A^{(n)})$ with FALSE and TRUE respectively to get an ordinary unweighted acceptor over the Boolean semiring; determinize this by standard methods; and then replace all FALSE and TRUE weights with $\bar{0}$ and $\bar{1}$ respectively.

```
AutoIntersectSinglePair(A_1^(n), i, j, δ^max_⟨i,j⟩) → A^(n) :
1   A^(n) ← ⟨Σ← Σ_1, Q← ∅, K← K_1, E^(n)← ∅, λ, ρ⟩
2   Stack ← ∅
3   for ∀q_1 ∈ {Q_1 : λ(q_1) ≠ 0̄} do
4       getPushState(q_1, (ε, ε))
5   while Stack ≠ ∅ do
6       q ← pop(Stack)
7       q_1 ← ν[q]
8       (s, u) ← ξ[q]
9       for ∀e_1 ∈ E(q_1) do
10          (s', u') ← createLeftoverStrings( s · π_⟨i⟩(ℓ(e_1)), u · π_⟨j⟩(ℓ(e_1)))
11          if (s'=ε ∨ u'=ε) ∧ ( |(|s'| − |u'|)| ≤ δ^max_⟨i,j⟩(A_1^(n)) )
12              then q' ← getPushState( n(e_1), (s', u'))
13                   E ← E ∪ { ⟨q, ℓ(e_1), w(e_1), q'⟩ }
14  return A^(n)

createLeftoverStrings(ṡ, u̇) → (s', u') :
15  x ← longestCommonPrefix(ṡ, u̇)
16  return (x^{-1} · ṡ, x^{-1} · u̇)

getPushState(q_1, (s', u')) → q' :
17  if ∃q ∈ Q : ν[q] = q_1 ∧ ξ[q] = (s', u')
18      then q' ← q
19      else q' ← createNewState( )
20           ν[q'] ← q_1
21           ξ[q'] ← (s', u')
22           if s' = ε ∧ u' = ε
23               then λ(q') ← λ(q_1)
24                    ρ(q') ← ρ(q_1)
25               else λ(q') ← 0̄
26                    ρ(q') ← 0̄
27           Q ← Q ∪ {q'}
28           push(Stack, q')
29  return q'
```

Fig. 5. The main algorithm AutoIntersectSinglePair. It relies on a prior computation of $\delta^{\max}_{\langle i,j\rangle}(A_1^{(n)})$.

and put $A'^{(n)} = (A^{(n)} \bowtie_{\{i=1\}} A^{(1)}_j) \bowtie_{\{j=1\}} A^{(1)}_i$.[9] Now $\sigma_{\{i=j\}}(A)$ can be found as $\sigma_{\{i=j\}}(A')$, which helps if $\langle A', i, j\rangle$ falls in Θ.

The second point is related to the generalization (7) for auto-intersection on multiple pairs of tapes. Given a problem $\sigma_{\{i_1=j_1,\ldots i_r=j_r\}}(A)$, we nondeterministically select a pair (i_h, j_h) (if any) such that $\langle A, i_h, j_h\rangle \in \Theta$, and use our method to compute $A' = \sigma_{\{i_h=j_h\}}(A)$. We now attempt to continue in the same way by auto-intersecting A' on the remaining $r-1$ tapes. Note that A' may have fewer cycles than A, so we may have $\langle A', i_{h'}, j_{h'}\rangle \in \Theta$ even if $\langle A, i_{h'}, j_{h'}\rangle \notin \Theta$.

Acknowledgments

We wish to thank 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 advice.

References

1. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development **3** (1959) 114–125
2. Elgot, C.C., Mezei, J.E.: On relations defined by generalized finite automata. IBM Journal of Research and Development **9** (1965) 47–68
3. Kay, M.: Nonconcatenative finite-state morphology. In: Proc. 3rd Int. Conf. EACL, Copenhagen, Denmark (1987) 2–10
4. Harju, T., Karhumäki, J.: The equivalence problem of multitape finite automata. Theoretical Computer Science **78** (1991) 347–355
5. Kaplan, R.M., Kay, M.: Regular models of phonological rule systems. Computational Linguistics **20** (1994) 331–378
6. Kiraz, G.A.: Multitiered nonlinear morphology using multitape finite automata: a case study on Syriac and Arabic. Computational Lingistics **26** (2000) 77–105
7. Kempe, A., Champarnaud, J.M., Eisner, J.: A note on join and auto-intersection of n-ary rational relations. In Watson, B., Cleophas, L., eds.: Proc. Eindhoven FASTAR Days. Number 04–40 in TU/e CS TR, Eindhoven, Netherlands (2004) 64–78
8. Frougny, C., Sakarovitch, J.: Synchronized rational relations of finite and infinite words. Theoretical Computer Science **108** (1993) 45–82
9. Mohri, M.: Edit-distance of weighted automata. In: Proc. 7th Int. Conf. CIAA (2002). Volume 2608 of Lecture Notes in Computer Science., Tours, France, Springer Verlag, Berlin, Germany (2003) 1–23
10. Eilenberg, S.: Automata, Languages, and Machines. Volume A. Academic Press, San Diego (1974)
11. Kuich, W., Salomaa, A.: Semirings, Automata, Languages. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer Verlag, Berlin, Germany (1986)
12. Mohri, M., Pereira, F.C.N., Riley, M.: A rational design for a weighted finite-state transducer library. Lecture Notes in Computer Science **1436** (1998) 144–158

[9] These single-tape joins are guaranteed to succeed (for commutative semirings): they can be computed similarly to transducer composition.

13. Kempe, A., Guingne, F., Nicart, F.: Algorithms for weighted multi-tape automata. Research report 2004/031, Xerox Research Centre Europe, Meylan, France (2004)
14. Rosenberg, A.L.: On n-tape finite state acceptors. In: IEEE Symposium on Foundations of Computer Science (FOCS). (1964) 76–81
15. Eisner, J.: Parameter estimation for probabilistic finite-state transducers. In: Proc. of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia (2002)
16. Kempe, A.: NLP applications based on weighted multi-tape automata. In: Proc. 11th Conf. TALN, Fes, Morocco (2004) 253–258