Appeared in Loek Cleophas and Bruce Watson (eds.), *Proceedings of the Eindhoven FASTAR Days* (Comp. Sci. Tech. Report 04-40). Dept. of Mathematics and Comp. Sci., Technische Universiteit Eindhoven, Dec. 2004.

A Note on Join and Auto-Intersection of n-ary Rational Relations

Andre Kempe[1] Jean-Marc Champarnaud[2] Jason Eisner[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] Université de Rouen Faculté des Sciences et des Techniques
76821 Mont-Saint-Aignan – France
Jean-Marc.Champarnaud@univ-rouen.fr

[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/

Abstract

A finite-state machine with n tapes describes a rational (or regular) relation on n strings. It is more expressive than a relational database table with n columns, which can only describe a *finite* relation.

We describe some basic operations on n-ary rational relations and propose notation for them. (For generality we give the semiring-weighted case in which each tuple has a weight.) Unfortunately, the join operation is problematic: if two rational relations are joined on more than one tape, it can lead to non-rational relations with undecidable properties. We recast join in terms of "auto-intersection" and illustrate some cases in which difficulties arise. We close with the hope that partial or restricted algorithms may be found that are still powerful enough to have practical use.

1 Introduction

Multi-tape finite-state machines (FSMs) (Rabin and Scott, 1959; Elgot and Mezei, 1965; Kay, 1987; Kaplan and Kay, 1994) are a natural generalization of the familiar one- and two-tape cases, known respectively as finite-state acceptors and transducers.

An n-tape FSM characterizes n-tuples of strings. The set of tuples that it accepts is called an n-ary relation. If the FSM is weighted, it defines a weighted n-ary relation that assigns each n-tuple a weight (in some semiring), such as a probability.

The relations defined by FSMs are known as *rational* (or *regular*) relations. Our interest in n-tuples stems from our view of these relations as relational databases. In the familiar case $n = 2$, a finite-state transducer can be regarded as a kind of (weighted) database of string pairs—for example, $\langle$spelling, pronunciation$\rangle$, $\langle$French word, English word$\rangle$, or $\langle$parent concept, child concept$\rangle$. An acyclic transducer can represent any finite database of this sort. Shared substrings can make the representation particularly efficient: a hypothesis lattice for speech processing (Mohri, 1997) represents exponentially many pairs in linear space.

Unlike a classical database, a transducer may even define infinitely many pairs. For example, it may characterize the pattern of the spelling-pronunciation relationship in such a way that it can map even a novel word's spelling to zero or more possible pronunciations (with various weights), and vice-versa. Another transducer may attempt to map not just a word but a sentence of unbounded length to an annotated, corrected, or translated version.

On this database view, it is natural to consider relations with more than 2 columns. In natural language processing, multi-tape machines have recently been used to represent lattices of $\langle$speech, gesture, interpretation$\rangle$ triples for processing multimodal input (Bangalore and Johnston, 2000). They have also been used in the morphological analysis of Semitic languages, using multiple tapes to synchronize the vowels, consonants, and templatic pattern into a surface form (Kay, 1987; Kiraz, 2000). They may be similarly useful for coordinating the multiple tiers of autosegmental phonology or articulator-based speech recognition (Livescu, Glass, and Bilmes, 2003).

Unfortunately, one pays a price for allowing infinite multi-column databases. Finite-state methods derive their power from a *rational* algebra, which can combine simple FSMs using operations such as union, closure, and composition. Databases similarly derive their power from a *relational* algebra. Cyclic FSMs are closed under the rational operations, but not under the relational operations, as finite databases are. For example, transducers are not closed under intersection (Rabin and Scott, 1959).

In this paper, we give a formal discussion of semiring-weighted n-ary relations (Section 2). We define several useful operators (Section 3), offering useful notation and taking care to distinguish cases that preserve the rationality of relations from those that do not.

The focus of the paper is a database join operator $\bowtie$ that generalizes intersection, composition, and cross product (Section 3.3). Certain cases of join (single-tape or finite) are guaranteed to preserve rationality and appear practically useful.

In Section 3.4, we reduce the join problem to a somewhat simpler problem of "auto-intersection" (Kempe, Guingne, and Nicart, 2004). In Section 4, we illustrate how auto-intersecting two tapes of a rational relation may produce a variety of non-rational weighted or unweighted relations, including context-sensitive languages whose emptiness is undecidable. We leave open the possibility that there may exist a partial or approximate algorithm with enough coverage to have some practical use.

2 Definitions

After recalling the basic definitions of a monoid and a semiring, we define n-ary weighted relations and n-tape weighted finite-state machines. Our definitions follow the usual definitions for multi-tape finite-state automata (Elgot and Mezei, 1965; Eilenberg, 1974), with semiring weights added just as for acceptors and transducers (Kuich and Salomaa, 1986; Mohri, Pereira, and Riley, 1998).

2.1 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 *semiring* is a structure $\mathcal{K} = \langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$ consisting of a set $\mathbb{K}$, two binary operations, $\oplus$ (*collection*) and $\otimes$ (*extension*), and two neutral elements, $\bar{0}$ and $\bar{1}$, that satisfies the following properties:

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

- $\bar{0}$ is an annihilator for extension: $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}\,, \quad \forall a \in \mathbb{K}$

Examples of semirings are:

1. $\langle \{\text{FALSE}, \text{TRUE}\}, \vee, \wedge, \text{FALSE}, \text{TRUE} \rangle$: the boolean semiring, which can be used to define unweighted relations and machines.
2. $\langle \mathbb{N}, +, \times, 0, 1 \rangle$: a non-negative integer semiring.
3. $\langle \mathbb{R}_{\geq 0}, +, \times, 0, 1 \rangle$: a non-negative real semiring that can be used to model probabilities.
4. $\langle \mathbb{R}_{\geq 0} \cup \{\infty\}, \min, +, \infty, 0 \rangle$: a "tropical" semiring, sometimes used to model negative logarithms of probabilities.
5. $\langle 2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\} \rangle$: the semiring of unweighted languages over an alphabet Σ under union $\cup$ and pairwise concatenation $\cdot$. Note that this has a subsemiring consisting of only the regular languages. (Similar semirings exist whose elements are weighted languages and relations, but we do not define them here.)

A semiring can have additional properties, and in this article we are interested in the following two:

1. commutativity: $a \otimes b = b \otimes a\,, \quad \forall a, b \in \mathbb{K}$
2. idempotency: $a \oplus a = a\,, \quad \forall a \in \mathbb{K}$

All examples above are commutative, except the last one, which is commutative only if $|\Sigma| = 1$. Examples 1, 4, and 5 are idempotent.

We will use the following notations for repeated collection and extension of a single value $k \in \mathbb{K}$:

$$ik \;=\; k \oplus k \oplus \;\cdots\; \oplus k \qquad (i \text{ times}) \tag{1}$$

$$k^i \;=\; k \otimes k \otimes \;\cdots\; \otimes k \qquad (i \text{ times}) \tag{2}$$

Note that ik does not in general mean $i \otimes k$. Usually the latter is not even defined, as the integer $i \in \mathbb{N}$ is usually not an element of the semiring.

2.2 Weighted n-ary Relations and Multi-Tape Weighted Finite-State Machines

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 $\mathcal{K} = \langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$. In other words, the 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.

We are especially interested in *rational* (or *regular*) n-ary relations—that is, relations that can be encoded by n-tape weighted finite-state machines, which we now define.

We adopt a 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 more complex objects that contain n-tuples of strings, such as n-tape automata and their transitions and paths.

An *n-tape weighted finite-state machine* (WFSM or n-WFSM),[1] $A^{(n)}$, is defined by a six-tuple

$$A^{(n)} = \langle \Sigma, Q, \mathcal{K}, E^{(n)}, \lambda, \varrho \rangle \tag{3}$$

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, $\lambda : Q \rightarrow \mathbb{K}$ a function that assigns initial weights to states, and $\varrho : Q \rightarrow \mathbb{K}$ a function that assigns final weights to states.

[1] We follow some recent literature in using the term "machine" rather than "automaton." The acronym to refer to the general n-tape case is then FSM or n-FSM, which leaves the acronym FSA available to refer to the special case of a finite-state *acceptor* ($n = 1$). FST refers to the special case of a finite-state transducer ($n = 2$).

Any transition $e^{(n)} \in E^{(n)}$ has the form

$$e^{(n)} = \langle p, \ell^{(n)}, w, n \rangle \tag{4}$$

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 $n(e^{(n)}) \in Q$.

A *path* $\gamma^{(n)}$ of length $\ell \geq 0$ is a sequence of transitions $e_1^{(n)} e_2^{(n)} \cdots e_\ell^{(n)}$ such that $n(e_i^{(n)}) = p(e_{i+1}^{(n)})$ for each $i \in [\![1, \ell - 1]\!]$. A path's label is defined to be the elementwise concatenation of the labels of its transitions:

$$\ell(\gamma^{(n)}) \stackrel{\text{def}}{=} \ell(e_1^{(n)}) \cdot \ell(e_2^{(n)}) \cdot \cdots \cdot \ell(e_\ell^{(n)}) \tag{5}$$

This is an n-tuple of strings having the form $s^{(n)} = \langle s_1, s_2, \ldots, s_n \rangle$. The path's weight is defined to be

$$w(\gamma^{(n)}) \stackrel{\text{def}}{=} \lambda(p(e_1^{(n)})) \otimes \left(\bigotimes_{j \in [\![1,\ell]\!]} w\left(e_j^{(n)}\right) \right) \otimes \varrho(n\left(e_\ell^{(n)}\right)) \tag{6}$$

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 $s^{(n)}$:

$$\Gamma_{A^{(n)}}(s^{(n)}) = \{ \gamma^{(n)} \in \Gamma_{A^{(n)}} \mid s^{(n)} = \ell(\gamma^{(n)}) \} \tag{7}$$

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:

$$\mathcal{R}_{A^{(n)}}(s^{(n)}) \stackrel{\text{def}}{=} \bigoplus_{\gamma^{(n)} \in \Gamma_{A^{(n)}}(s^{(n)})} w(\gamma^{(n)}) \tag{8}$$

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

$$\mathsf{support}(\mathcal{R}^{(n)}) \stackrel{\text{def}}{=} \{ s^{(n)} \in (\Sigma^*)^n \mid \mathcal{R}^{(n)}(s^{(n)}) \neq \bar{0} \} \tag{9}$$

This support set can be regarded as an ordinary unweighted relation obtained from $\mathcal{R}^{(n)}$. A different perspective on unweighted relations is that they are weighted relations over the boolean semiring, i.e., functions from $(\Sigma^*)^n \to \{\text{FALSE}, \text{TRUE}\}$.

2.3 Infinite Sums

In defining $\mathcal{R}(A^{(n)})$, we glossed over one point for simplicity's sake. A sum over finitely many weights can be computed by repeated application of $\oplus$. But (8) may sometimes call for an infinite sum, whose meaning has not been defined. This case arises if $\mathcal{R}_{A^{(n)}}$ contains any cyclic paths with the label $\langle \epsilon, \epsilon, \ldots \epsilon \rangle$. Cyclic paths of this sort cannot simply be disallowed in a natural way, since they can be re-introduced by the closure and projection operations discussed below.

Briefly, the solution is to pre-compute the geometric sum $k^* = \bigoplus_{i=0}^{\infty} k^i \in \mathbb{K}$ for each $k \in \mathbb{K}$.[2] In practice, one simply defines a *closure* operator * that satisfies certain axioms, obtaining a so-called *closed semiring*.

This allows infinite sums over any *regular* set of paths, as required by (8) and by section 3's equations (11), (12), (13), and (21). One constructs a WFSM containing just those paths (e.g., $\Gamma_{A^{(n)}}(s^{(n)})$), and then sums their weights with an algorithm that generalizes the Kleene-Floyd-Warshall technique to closed semirings (Lehmann, 1977).

[2] Divergent sums can be represented by $k^* = \infty$, where $\infty \in \mathbb{K}$ is a distinguished value.

3 Operations

We now describe some central operations on n-ary weighted relations and their n-tape WFSMs, focusing on operations that affect the number of tapes. (See (Kempe, Guingne, and Nicart, 2004).) In particular, we introduce an "auto-intersection" operation that will simplify the discussion of multi-tape join.

Our notation is chosen throughout to highlight the connection to relational databases.

3.1 Simple Operations

The basic rational operations of union, concatenation, and closure can be used to construct any n-ary weighted rational relation.[3] Thus, the rational operations can be used to write *regular expressions* that specify particular relations. On the database perspective, such expressions are useful for specifying both actual databases (typically finite relations) and particular queries (typically infinite relations, i.e., the set of all tuples with a given property). (Section 3.3 will discuss how to intersect a database with a query.)

The union and concatenation of two weighted n-ary relations, $\mathcal{R}_1^{(n)}$ and $\mathcal{R}_2^{(n)}$, are the relations $\mathcal{R}_1^{(n)} \cup \mathcal{R}_2^{(n)}$ and $\mathcal{R}_1^{(n)} \cdot \mathcal{R}_2^{(n)}$ defined by

$$\left(\mathcal{R}_1^{(n)} \cup \mathcal{R}_2^{(n)}\right)(s^{(n)}) \stackrel{\text{def}}{=} \mathcal{R}_1^{(n)}(s^{(n)}) \oplus \mathcal{R}_2^{(n)}(s^{(n)}) \tag{10}$$

$$\left(\mathcal{R}_1^{(n)} \cdot \mathcal{R}_2^{(n)}\right)(s^{(n)}) \stackrel{\text{def}}{=} \bigoplus_{\substack{u^{(n)},v^{(n)}:\\ (\forall i \in [\![1,n]\!]) s_i = u_i \cdot v_i}} \mathcal{R}_1^{(n)}(u^{(n)}) \otimes \mathcal{R}_2^{(n)}(v^{(n)}) \tag{11}$$

The closure of $\mathcal{R}^{(n)}$ is the relation

$$(\mathcal{R}^{(n)})^* \stackrel{\text{def}}{=} \bigcup_{\ell=0}^{\infty} \underbrace{\mathcal{R}^{(n)} \cdot \mathcal{R}^{(n)} \cdots \mathcal{R}^{(n)}}_{\ell \text{ times}}, \quad \text{implying that}$$

$$\left((\mathcal{R}^{(n)})^*\right)(\langle s_1, \ldots, s_n \rangle) = \bigoplus_{\ell=0}^{\infty} \bigoplus_{\substack{u_1^{(n)},\ldots u_\ell^{(n)}:\\ (\forall i \in [\![1,n]\!]) s_i = (u_1)_i \cdot (u_2)_i \cdots (u_\ell)_i}} \bigotimes_{j=1}^{\ell} \mathcal{R}^{(n)}(u_j^{(n)}) \tag{12}$$

These operations can be implemented by simple constructions on the corresponding nondeterministic n-tape WFSMs (Rosenberg, 1964). These n-tape constructions and their semiring-weighted versions are exactly the same as for acceptors ($n = 1$) and transducers ($n = 2$), as they are indifferent to the n-tuple transition labels.

3.2 Projection and Complementary Projection

Projection keeps certain columns of a database relation and discards the others. In the case of a rational relation implemented by a n-WFSM, it can be implemented by discarding the corresponding tapes of the n-WFSM, yielding an m-WFSM for $m < n$.

Projection may map several distinct n-tuples onto the same m-tuple. In this case, we will define the weight of the m-tuple by summing the several n-tuples' weights using $\oplus$. This resembles aggregation in databases, but note that only weights can be aggregated across n-tuples, not the (string) data in the n-tuples themselves.

[3] By combining the "atomic" weighted relations, namely, those whose support is a single tuple from the finite set $\{(s_1, s_2, \ldots s_n) : |s_1 s_2 \cdots s_n| \leq 1\}$.

For any $j_1, \ldots, j_m \in [\![1, n]\!]$, we formally define a *projection* operator $\pi_{\langle j_1,\ldots,j_m \rangle}$ that maps n-ary relations to m-ary relations:

$$\left(\pi_{\langle j_1,\ldots,j_m\rangle}(\mathcal{R}_1^{(n)})\right)(s^{(m)}) \quad \stackrel{\text{def}}{=} \quad \bigoplus_{\substack{u^{(n)}:\\ (\forall i \in [\![1,m]\!])\, s_i = u_{j_i}}} \mathcal{R}_1^{(n)}(u^{(n)}) \tag{13}$$

It retains only those component strings (i.e. tapes) of each tuple that are specified by the indices $j_1, \ldots j_m$, and places them in the specified order.

Notice that our definition allows projection indices to occur in any order, possibly with repeats. Thus the tapes of $s^{(n)}$ can be permuted or duplicated. For example, $\pi_{\langle 2,1\rangle}$ will invert a 2-ary relation.

As a convenience, we also define the *complementary projection* of a relation. For any $j_1, \ldots j_m \in [\![1, n]\!]$, we define an operator $\overline{\pi}_{\{j_1,\ldots j_m\}}$ that removes the tapes $j_1, \ldots j_m$ and preserves all other tapes in their original order. Without loss of generality we may assume that $j_1 < j_2 < \cdots < j_m$; then we can define $\overline{\pi}_{\{j_1,\ldots j_m\}}$ as equivalent to $\pi_{\langle 1,\ldots,j_1-1,j_1+1,\ldots j_m-1,j_m+1,\ldots n\rangle}$, which maps n-ary relations to $(n-m)$-ary relations.

3.3 Join and Generalized Composition

Applications: Our version of the join operation is quite powerful. It can be used to join two "databases" (typically finite relations), to conjoin two "queries" (typically infinite relations), or to select those database tuples that match a query, reweighting them if the query is weighted.

Another family of uses is inspired by natural language processing, where WFSTs ($n = 2$) are commonly used to construct noisy channel models (Knight and Graehl, 1998). Using $n > 2$ tapes allows us to generalize naturally to doing constraint programming or graphical modeling over string-valued variables. Given variables $V_1, \ldots V_n$ with unknown values in the *infinite* domain Σ^*, one can specify a (weighted) m-ary relation to express a (soft) constraint over some $m \leq n$ of the variables. All known constraint relations can be systematically joined together, along tapes that correspond to common variables. This yields a (weighted) n-ary relation that evaluates which n-tuples are appropriate as joint values of the n variables. If this n-ary relation specifies a probability distribution over n-tuples, one can intersect it with another n-ary relation describing incomplete data, in order to compute the probability of the data for purposes of parameter training or statistical inference.

As we will see, join is *too* powerful: rational relations are *not* closed under arbitrary joins. Section 4 will explore this point in detail. Nonetheless, we can mathematically define the possibly non-rational result of a join. The operation appears so useful that it would be helpful to have a partial or approximate algorithm.

Definition: The reader may already be familiar with the notion of natural join on databases. Our presentation differs from the standard database treatment in that our tapes are numbered, whereas the columns of a database are typically named. So our join operators, unlike a database join, must explicitly select tapes by number, and as a result are neither associative nor commutative.

A join of two relations is formed by finding "matching" pairs of tuples. For example, $\langle abc, def, \epsilon\rangle$ and $\langle def, ghi, \epsilon, jkl\rangle$ match on two of their tapes. We notate the matching of tapes in this case as $\{2{=}1, 3{=}3\}$. They combine to yield a tuple $\langle abc, def, \epsilon, ghi, jkl\rangle$, whose weight in the joined relation is the product (under $\otimes$) of the two original tuples' weights.

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

$$\left(\mathcal{R}_1^{(n)} \bowtie_{\{i_1=j_1,\ldots,i_r=j_r\}} \mathcal{R}_2^{(m)}\right)(\langle u_1,\ldots,u_n,s_1,\ldots,s_{m-r}\rangle) \stackrel{\text{def}}{=} \mathcal{R}_1^{(n)}(u^{(n)}) \otimes \mathcal{R}_2^{(m)}(v^{(m)}) \tag{14}$$

where $v^{(m)}$ is the unique tuple such that $\overline{\pi}_{\{j_1,\ldots j_r\}}(v^{(m)}) = s^{(m-r)}$ and $(\forall \ell \in [\![1,r]\!]) v_{j_\ell} = u_{i_\ell}$.

Relation to Cross Product: Taking $r = 0$ gives an important special case. The *cross product* operator $\times$, equivalent to $\bowtie_\emptyset$, combines an n-ary and an m-ary relation into an $(n+m)$-ary relation:

$$\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)} \stackrel{\text{def}}{=} \mathcal{R}_1^{(n)} \bowtie_\emptyset \mathcal{R}_2^{(m)} \tag{15}$$

with the result that

$$\left(\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)}\right)(\langle u_1,\ldots,u_n,v_1,\ldots,v_m\rangle) = \mathcal{R}_1^{(n)}(u^{(n)}) \otimes \mathcal{R}_2^{(m)}(v^{(m)}) \tag{16}$$

A WFSM for $\mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)}$ can easily be constructed from WFSMs for $\mathcal{R}_1^{(n)}$ and $\mathcal{R}_2^{(m)}$, by concatenating them after appropriately "padding" their transition labels into $(n+m)$-tuples via extra epsilons. Thus, the cross product of weighted rational relations is always rational.

Relation to Intersection: Taking $n = r = m$ gives another important special case. The *intersection* of two n-ary relations is another n-ary relation:

$$\mathcal{R}_1^{(n)} \cap \mathcal{R}_2^{(n)} \stackrel{\text{def}}{=} \mathcal{R}_1^{(n)} \bowtie_{\{1=1,2=2,\ldots n=n\}} \mathcal{R}_2^{(n)} \tag{17}$$

with the result that

$$\left(\mathcal{R}_1^{(n)} \cap \mathcal{R}_2^{(n)}\right)(s^{(n)}) = \mathcal{R}_1^{(n)}(s^{(n)}) \otimes \mathcal{R}_2^{(n)}(s^{(n)}) \tag{18}$$

It is known that the intersection of transducers ($n = 2$) is not necessarily rational (Rabin and Scott, 1959): $\{\langle a^j b^*, c^j\rangle \mid j \in \mathbb{N}\} \cap \{\langle a^* b^j, c^j\rangle \mid j \in \mathbb{N}\} = \{\langle a^j b^j, c^j\rangle \mid j \in \mathbb{N}\}$. Nor, for that matter, is intersection of acceptors ($n = 1$) if they are weighted by a non-commutative semiring. Thus rational relations are not closed under the more general join operation, either.

Generalized Composition: For distinct $i_1,\ldots i_r \in [\![1,n]\!]$ and distinct $j_1,\ldots j_r \in [\![1,m]\!]$, it is convenient to define a *generalized composition* operator $\boxtimes_{\{i_1=j_1,\ldots,i_r=j_r\}}$. It carries out a join and then discards the joined tapes:

$$\mathcal{R}_1^{(n)} \boxtimes_{\{i_1=j_1,\ldots,i_r=j_r\}} \mathcal{R}_2^{(m)} \stackrel{\text{def}}{=} \overline{\pi}_{\{i_1,\ldots i_r\}}\left(\mathcal{R}_1^{(n)} \bowtie_{\{i_1=j_1,\ldots,i_r=j_r\}} \mathcal{R}_2^{(m)}\right) \tag{19}$$

Note that $\boxtimes$ can result in aggregation because it uses $\overline{\pi}$. For example, the special case of ordinary composition $\circ$ of transducers

$$\mathcal{R}_1^{(2)} \circ \mathcal{R}_2^{(2)} \stackrel{\text{def}}{=} \mathcal{R}_1^{(2)} \boxtimes_{\{2=1\}} \mathcal{R}_2^{(2)} = \overline{\pi}_{\{2\}}(\mathcal{R}_1^{(2)} \bowtie_{\{2=1\}} \mathcal{R}_2^{(2)}) \tag{20}$$

results in a summation over strings v on the discarded tape that was joined:

$$\left(\mathcal{R}_1^{(2)} \circ \mathcal{R}_2^{(2)}\right)(u,w) = \bigoplus_v \mathcal{R}_1^{(2)}(u,v) \otimes \mathcal{R}_2^{(2)}(v,w) \tag{21}$$

The generalized composition of rational relations is not necessarily rational.

Single-Tape Join: We speak about *single-tape join* if only one tape is used in each relation ($r = 1$). Two well-known special cases are the join $\bowtie_{\{1=1\}}$ used to intersect two acceptors in (17) (where $n = 1$), and the join $\bowtie_{\{2=1\}}$ used during classical composition of two transducers in (20).

There are other uses of single-tape join. A composition cascade of several transducers, $\mathcal{R}^{(2)} = \mathcal{R}_1^{(2)} \circ \mathcal{R}_2^{(2)} \circ \mathcal{R}_3^{(2)}$, could be replaced by a join cascade, $\mathcal{R}^{(4)} = \mathcal{R}_1^{(2)} \bowtie_{\{2=1\}} \left(\mathcal{R}_2^{(2)} \bowtie_{\{2=1\}} \mathcal{R}_3^{(2)}\right)$. The intermediate results are now preserved on tapes 2 and 3 for subsequent inspection or further transduction (Kempe, 2004). In this way, single-tape join is adequate to combine several transducers into any tree topology: $\mathcal{R}^{(4)} = \left(\mathcal{R}_1^{(2)} \bowtie_{\{2=1\}} \mathcal{R}_2^{(2)}\right) \bowtie_{\{2=1\}} \mathcal{R}_3^{(2)}$. One can use this technique to implement a tree-structured directed graphical model (sometimes called a dendroid distribution) by joining weighted transducers that represent the conditional probability distributions of the model.

Sometimes one wishes to join an n-ary relation with a cross product of m languages. This operation can be regarded as m single-tape joins. It can be used to train the parameters of the dendroid distribution described above, as explained for $n = m = 2$ by (Eisner, 2002). The generalization to more tapes is particularly useful for training a cascaded noisy channel model when intermediate results along the channel are partly observed.

The single-tape join of weighted multi-tape rational relations is rational as long as the weights fall in a commutative weight semiring. One can construct a WFSM for the resulting relation, using a standard "cross-product of states" construction.

The commutativity of the weights is crucial to this construction. (The constructed WFSM's paths interleave weights from paths in the two input WFSMs.) No such construction is possible if the weight semiring $\mathcal{K}$ is not commutative. For example, let k, k' be weights that do not commute. Let $\mathcal{R}^{(1)}$ be a rational language such that $\forall j \in \mathbb{N}$, $\mathcal{R}^{(1)}(a^j) = k^j \otimes k'$. Then (18) implies that $\forall j$, $\left(\mathcal{R}^{(1)} \cap \mathcal{R}^{(1)}\right)(a^j) = k^j \otimes k' \otimes k^j \otimes k'$; this single-tape join cannot in general be computed by any WFSM.

Mohri, Pereira, and Riley (1998), writing about WFST composition, noted another subtlety in extending the "cross product of states" construction to weighted machines. Their observation and solution apply generally to single-tape join of WFSMs (and would presumably be relevant to any partial algorithm for multi-tape join). A pair of successful paths in the input machines are considered to "match" if they both accept the same string s on the single tape being joined. A pair of matched input paths is supposed to yield exactly one path in the composed machine. However, if both input paths allow ϵ transitions on the join tape at the same position in s, then a naive implementation of the construction may produce $i > 1$ identically labeled and weighted paths, corresponding to different alignments of the input paths. This "path multiplicity problem" will incorrectly contribute i copies of the path weight to the sum in (8), affecting the result unless the weight semiring is idempotent. The solution is to revise the construction to allow only a canonical alignment of matched input paths.

3.4 Auto-Intersection

Our discussion of join will be simplified by reducing it to a simpler problem. For any distinct $i_1, j_1, \ldots i_r, j_r \in [\![1, n]\!]$, we define an *auto-intersection* operator $\sigma_{\{i_1=j_1, i_2=j_2, \ldots, i_r=j_r\}}$ that 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 all other tuples from the support of the relation.[4]

$$\left(\sigma_{\{i_1=j_1,\ldots,i_r=j_r\}}(\mathcal{R}^{(n)})\right)(\langle s_1, \ldots, s_n\rangle) \stackrel{\text{def}}{=} \begin{cases} \mathcal{R}^{(n)}(\langle s_1, \ldots, s_n\rangle) & \text{if } (\forall \ell \in [\![1,r]\!]) s_{i_\ell} = s_{j_\ell} \\ \bar{0} & \text{otherwise} \end{cases} \quad (22)$$

Auto-intersection does not necessarily preserve the rationality of $\mathcal{R}^{(n)}$, as we will discuss in Section 4.

[4] The requirement that the $2r$ indices be distinct mirrors the similar requirement on join and is needed in (26). But it can be evaded by duplicating tapes: an illegal auto-intersection such as $\sigma_{\{1=2,2=3\}}(\mathcal{R})$ can be computed as $\overline{\pi}_{\{3\}}(\sigma_{\{1=2,3=4\}}(\pi_{\langle 1,2,2,3\rangle}(\mathcal{R})))$.

Note that auto-intersecting a relation is different from joining the relation 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, $\mathcal{R}^{(2)} \bowtie_{\{1=1\}} \big(\pi_{\langle 2\rangle}(\mathcal{R}^{(2)})\big)$ 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).

An example of auto-intersection is shown in Figure 1. It encodes the relation

$$\mathcal{R}_1^{(3)} = \langle a, x, \varepsilon\rangle\ \langle b, y, a\rangle^*\ \langle \varepsilon, z, b\rangle = \{\ \langle ab^j, xy^j z, a^j b\rangle \mid j \in \mathbb{N}\ \} \tag{23}$$

$$\sigma_{\{1=3\}}(\mathcal{R}_1^{(3)}) = \{\ \langle ab^1, xy^1 z, a^1 b\rangle\ \} \tag{24}$$

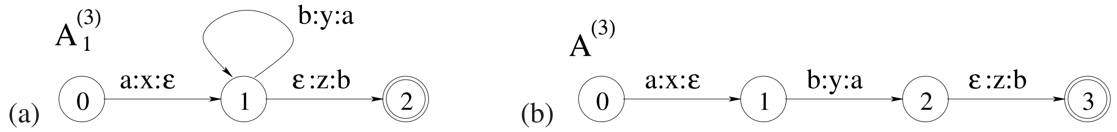

Figure 1: (a) A WFSM $A_1^{(3)}$ and (b) its auto-intersection $A^{(3)} = \sigma_{\{1=3\}}(A_1^{(3)})$. (Weights omitted)

It is possible to reduce join to auto-intersection using only rational operations (namely cross product and complementary projection). An arbitrary join can be implemented as

$$\mathcal{R}_1^{(n)} \bowtie_{\{i_1=j_1,\ldots,i_r=j_r\}} \mathcal{R}_2^{(m)} = \overline{\pi}_{\{n+j_1,\ldots,n+j_r\}}\Big(\ \sigma_{\{i_1=n+j_1,\ldots,i_r=n+j_r\}}(\ \mathcal{R}_1^{(n)} \times \mathcal{R}_2^{(m)}\ )\Big) \tag{25}$$

Conversely, it is possible to reduce any auto-intersection to a single join with a rational relation:

$$\sigma_{\{i_1=j_1,\ldots,i_r=j_r\}}(\mathcal{R}^{(n)}) = \mathcal{R}^{(n)} \bowtie_{\{i_1=1,j_1=2,\ldots,i_r=2r-1,j_r=2r\}} \left(\underbrace{(\pi_{\langle 1,1\rangle}(\Sigma^*) \times \cdots \times \pi_{\langle 1,1\rangle}(\Sigma^*)}_{r \text{ times}}\right) \tag{26}$$

Thus, for any class of "difficult" join instances whose results are non-rational or have undecidable emptiness (see section 4.4), 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. In future work we hope to identify such a partial algorithm for auto-intersection.

The rest of this paper is therefore devoted to remarks on the auto-intersection problem only. Working in terms of auto-intersection rather than join will simplify our discussion. First, only one machine is involved. Second, in considering partial algorithms for auto-intersection, we do not have to worry about the order in which non-commutative weights from two joined machines are multiplied together, or the path multiplicity problem. Those issues have already been handled in the cross-product step of the join construction (25), and are not of further concern to the auto-intersection step.

For simplicity, we will focus on auto-intersections $\sigma_{\{i=j\}}$ that involve only a single pair of tapes. That is enough to expose the core difficulties. Indeed, the general case of auto-intersection can be defined in terms of this simple case:

$$\sigma_{\{i_1=j_1,\ldots,i_r=j_r\}}(\ \mathcal{R}^{(n)}\ ) \stackrel{\text{def}}{=} \sigma_{\{i_r=j_r\}}(\ \cdots \sigma_{\{i_1=j_1\}}(\ \mathcal{R}^{(n)}\ ) \cdots\ ) \tag{27}$$

Nonetheless, we caution that the general case might benefit from a more direct treatment. 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 (27) 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 Some Difficult Examples for Auto-Intersection

Some instances of auto-intersection are "easy." In particular, consider a finite relation (one with finite support, representable by an acyclic WFSM). Its auto-intersection is computable and is itself finite, since it just selects some tuples of the original relation. (Thus, by (25), $\mathcal{R}_1 \bowtie \mathcal{R}_2$ is finite if $\mathcal{R}_1$ or $\mathcal{R}_2$ is.) On such "easy" examples, the job of a good auto-intersection algorithm is merely to keep the resulting FSM small by preserving the sharing of substrings in the original FSM.

In this section, we will discuss some "difficult" classes of auto-intersection problems, where the result is non-rational or has undecidable properties. Each such class has a matching class of join problems, as discussed in section 3.4.

These difficulties imply that there is no general finite-state join algorithm. Nor is there an algorithm that produces the join whenever it is rational and returns an error code otherwise.

At the same time, the examples in this section may be instructive if one wishes to design a more limited join or auto-intersection algorithm that can succeed (exactly or approximately) on some practical cases. We leave such a task to future work.

4.1 Equal-Exponent Problem

Consider the unweighted binary relation $\mathcal{R}^{(2)} = \{\langle a^i b^j, a^j b^k \rangle \mid i,j,k \in \mathbb{N}\}$, interpreted as a weighted relation over the boolean semiring. The relation is rational because it can be encoded by a 2-FSM (Figure 2a). Its auto-intersection $\sigma_{\{1=2\}}(\mathcal{R}^{(2)}) = \{\langle a^i b^i, a^i b^i \rangle \mid i \in \mathbb{N}\}$ is, however, non-rational. Notice that the auto-intersection would in effect need to select just those paths in Figure 2a where all three cycles are traversed the same number of times.

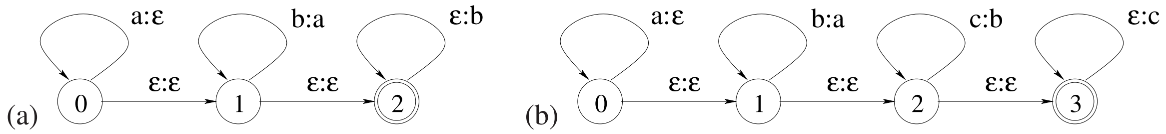

Figure 2: Two FSMs whose auto-intersection leads to equal-exponent problems

We can extend this example to any number of equal exponents. Consider for example the binary relation $\mathcal{R}^{(2)} = \{\langle a^i b^j c^k, a^j b^k c^\ell \rangle \mid i,j,k,\ell \in \mathbb{N}\}$, which is rational (Figure 2b) but has a non-rational auto-intersection $\sigma_{\{1=2\}}(\mathcal{R}^{(2)}) = \{\langle a^i b^i c^i, a^i b^i c^i \rangle \mid i \in \mathbb{N}\}$.

We say that such examples suffer from the *equal-exponent problem*. The equal-exponent problem may also appear on tapes other than the ones being intersected. The unweighted 3-ary relation $\{\langle a^i a, aa^j, x^i y z^j \rangle \mid i,j \in \mathbb{N}\}$ is rational (Figure 3a); but its auto-intersection under $\sigma_{\{1=2\}}$ is equal to $\{\langle a^i a, aa^i, x^i y z^i \rangle \mid i \in \mathbb{N}\}$, which is not rational because its projection onto tape 3 is not a regular language.

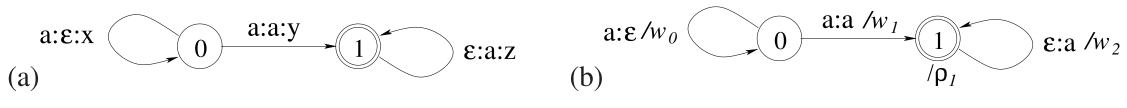

Figure 3: FSMs whose auto-intersection on tapes 1,2 requires equal exponents on tape 3 or in weights

Finally, the equal-exponent problem may appear in the weights assigned by the relation, if the weight semiring is not commutative. Figure 3b is a variant of Figure 3a that replaces the third tape with weights.

Its auto-intersection under $\sigma_{\{1=2\}}$ is the weighted relation $\mathcal{R}$ defined by

$$\mathcal{R}(\langle a^i a, aa^i \rangle) = w_0^i \otimes w_1 \otimes w_2^i \otimes \varrho_1 \tag{28}$$

$$\mathcal{R}(s^{(2)}) = \bar{0} \text{ otherwise} \tag{29}$$

This relation has rational support, but is not in general a rational relation. It does become rational if the weight semiring is commutative, in which case $w_0^i \otimes w_1 \otimes w_2^i \otimes \varrho_1$ can be computed as $(w_0 \otimes w_2)^i \otimes w_1 \otimes \varrho_1$. Notice that if the weights are rational languages over an alphabet Σ (see Section 2.1), so that they effectively act like a third tape, then they are guaranteed to commute only if $|\Sigma|$=1.

4.2 Shuffle Problem

The *shuffle product* of two strings $u \sqcup\!\sqcup\, v$ is defined, e.g., in (Sakarovitch, 2003) as:

$$u \sqcup\!\sqcup\, v \stackrel{\text{def}}{=} \{\, u_1 v_1 \ldots u_j v_j \mid u = u_1 \ldots u_j\,,\ v = v_1 \ldots v_j\,,\ (\forall i \in [\![1, j]\!]) u_i, v_i \in \Sigma^* \,\} \tag{30}$$

This set contains all possible "interleavings" of the symbols from u and v. The symbols of u keep their respective order, as do the symbols of v, but any order is allowed between a symbol from u and a symbol from v. For example:

$$abc \sqcup\!\sqcup\, xy = \{abcxy, abxcy, abxyc, axbcy, axbyc, axybc, xabcy, xabyc, xaybc, xyabc\} \tag{31}$$

$$aa \sqcup\!\sqcup\, xx = \{aaxx, axax, axxa, xaax, xaxa, xxaa\} \tag{32}$$

$$aaa \sqcup\!\sqcup\, aaa = \{aaaaaa\} \tag{33}$$

The size of the set $u \sqcup\!\sqcup\, v$ grows exponentially in the lengths of u and v.

Consider the unweighted relation $\mathcal{R}^{(3)} = \{\langle a^i, a^j, x^i \sqcup\!\sqcup\, y^j \rangle \mid i, j \in \mathbb{N}\}$, interpreted as a weighted relation over the boolean semiring. It is rational because it can be encoded by a 3-FSM (Figure 4a). Its auto-intersection $\sigma_{\{1=2\}}(\mathcal{R}^{(3)}) = \{\langle a^i, a^i, x^i \sqcup\!\sqcup\, y^i \rangle \mid i \in \mathbb{N}\}$ is, however, non-rational, as its projection onto tape 3 is the non-rational language of strings having equal numbers of x's and y's.

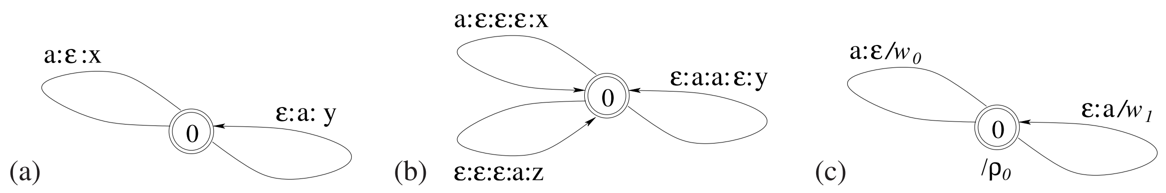

Figure 4: Three (W)FSMs whose auto-intersection leads to shuffle problems

Using additional tapes lets us extend this example to any number of equal exponents. For example, the relation $\mathcal{R}^{(5)} = \{\langle a^i, a^j, a^j, a^k, x^i \sqcup\!\sqcup\, y^j \sqcup\!\sqcup\, z^k \rangle \mid i, j, k \in \mathbb{N}\}$ is rational (Figure 2b) but has a non-rational auto-intersection $\sigma_{\{1=2,3=4\}}(\mathcal{R}^{(5)}) = \{\langle a^i, a^i, a^i, a^i, x^i \sqcup\!\sqcup\, y^i \sqcup\!\sqcup\, z^i \rangle \mid i \in \mathbb{N}\}$.

This *shuffle problem* can be regarded as the source of other failures of rationality. If $\mathcal{R}^{(1)}$ is any rational language, then the single-tape join $\{\langle a^i, a^j, (x^i \sqcup\!\sqcup\, y^j) \rangle \mid i, j \in \mathbb{N}\} \bowtie_{\{3=1\}} \mathcal{R}^{(1)}$ is also rational. Auto-intersecting it using the $\sigma_{\{1=2\}}$ operator yields a relation whose tape 3 recognizes a "restricted shuffle," namely, the potentially non-rational language $\{x^i \sqcup\!\sqcup\, y^i \mid i \in \mathbb{N}\} \cap \mathcal{R}^{(1)}$. For example, taking $\mathcal{R}^{(1)}$ to be the language $x^* y^*$ creates the equal-exponent language $\{x^i y^i \mid i \in \mathbb{N}\}$ of section 4.1.

Beyond simply restricting the shuffle language, one can also transduce it to obtain further examples. Consider the rational 3-relation $\{\langle a^i, a^j, (x^i \sqcup\!\sqcup\, y^j) \rangle \mid i, j \in \mathbb{N}\} \boxtimes_{\{3=1\}} \mathcal{R}^{(2)}$, where $\mathcal{R}^{(2)}$ is any rational

2-ary relation. Applying the $\sigma_{\{1=2\}}$ operator yields a relation whose tape 3 recognizes the transduction of $\{x^i \sqcup\!\sqcup\, y^i \mid i \in \mathbb{N}\}$ by $\mathcal{R}^{(2)}$. The transduction can replace x^i and y^i by arbitrary languages while restricting their shuffling.

The shuffle problem may also appear in the weights assigned by the relation, if the weight semiring is not both commutative and idempotent. Figure 4c is a variant of Figure 4a that replaces the third tape with weights.[5] Applying the $\sigma_{\{1=2\}}$ operator yields a relation $\mathcal{R}$ such that $\forall i \in \mathbb{N}$, $\mathcal{R}(a^i) = (w_0^i \sqcup\!\sqcup\, w_1^i) \otimes \varrho_0$, where the informal notation $w_0^i \sqcup\!\sqcup\, w_1^i$ denotes the "shuffle sum of two products of weights." For example, if $k, l, p, q \in \mathbb{K}$, we would write

$$
\begin{aligned}
(k \otimes l) \sqcup\!\sqcup\, (p \otimes q) &= (k \otimes l \otimes p \otimes q) \oplus (k \otimes p \otimes l \otimes q) \oplus (k \otimes p \otimes q \otimes l) \oplus \\
&\quad (p \otimes k \otimes l \otimes q) \oplus (p \otimes k \otimes q \otimes l) \oplus (p \otimes q \otimes k \otimes l) & (34) \\
k^2 \sqcup\!\sqcup\, p^2 &= (k \otimes k \otimes p \otimes p) \oplus (k \otimes p \otimes k \otimes p) \oplus (k \otimes p \otimes p \otimes k) \oplus \\
&\quad (p \otimes k \otimes k \otimes p) \oplus (p \otimes k \otimes p \otimes k) \oplus (p \otimes p \otimes k \otimes k) & (35) \\
k^2 \sqcup\!\sqcup\, k^2 &= 6\,(k \otimes k \otimes k \otimes k) \;=\; 6\,(\,k^4\,) & (36)
\end{aligned}
$$

In general, the weighted relation $\mathcal{R}$ in our example is non-rational. However, it is rational if the semiring is both commutative and idempotent. In that case, $w_0^i \sqcup\!\sqcup\, w_1^i = j_i(w_0 \otimes w_1)^i = (w_0 \otimes w_1)^i$, where $j_i \in \mathbb{N}$ is the number of summands in the shuffle sum and is irrelevant thanks to idempotency.

4.3 Presentation Problems

Our next example illustrates how a partial auto-intersection algorithm might be affected by the presentation of its input.

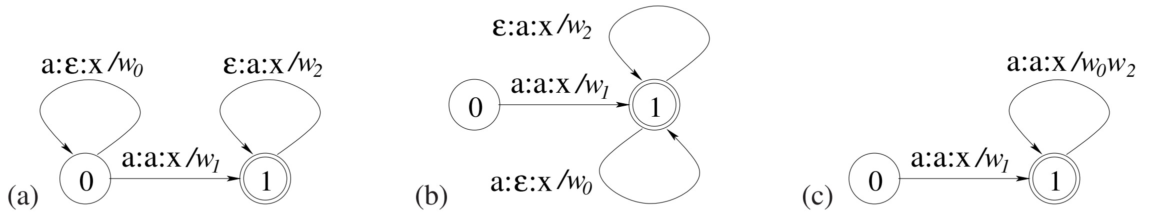

Figure 5: (a), (b) Different presentations of the same relation $\mathcal{R}^{(3)}$; (c) the auto-intersection $\sigma_{\{1,2\}}(\mathcal{R}^{(3)})$

Provided that the weight semiring is commutative, the WFSMs in Figures 4.3a and 4.3b describe the same relation, which for each $i \in \mathbb{N}$ maps $\langle a^{i+1}, a^{i+1}, x^{2i+1} \rangle$ to $w_0^i \otimes w_1 \otimes w_2^i$. A naive algorithm modeled on WFST determinization would fail to terminate on either machine, constructing a successful path of length $2i+1$ for each $i \in \mathbb{N}$. For example, on Figure 4.3a, it would allow unrolling the first cycle i times and then transitioning to the second cycle to allow the second tape to "catch up" with the first.

A partial algorithm for auto-intersection might attempt to detect and handle some such cases, allowing it to compute the correct auto-intersection (Figure 4.3c). It seems potentially easier to detect the Figure 4.3b case than the Figure 4.3a case.

[5] Again, this example can be derived by transducing the original shuffle example of Figure 4a. If all transitions in that example are given weight $\bar{1}$ in the semiring of interest, then its generalized composition $\boxtimes_{\{3=1\}}$ with a simple weighted machine will produce Figure 4c by replacing all instances of x with w_0, etc.

4.4 Post's Correspondence Problem

Post's Correspondence Problem or PCP (Post, 1946) is a classical undecidable problem that is sometimes used to prove the undecidability of other problems. Mark-Jan Nederhof (personal communication) pointed out its relevance to auto-intersection.

Definition: Given an alphabet Σ, an instance of PCP is a list of pairs of strings in Σ^*: $\langle u_1, v_1\rangle, \ldots \langle u_p, v_p\rangle$. A solution is a string s such that $s = u_{i_1} u_{i_2} \ldots u_{i_r} = v_{i_1} v_{i_2} \ldots v_{i_r}$ for some non-empty index sequence $i_1, i_2, \ldots i_r \in [\![1, p]\!]$. This sequence may contain duplicates.

Taking an example from (Zhao, 2002), the instance $\langle \texttt{abb}, \texttt{a}\rangle$, $\langle \texttt{b}, \texttt{abb}\rangle$, $\langle \texttt{a}, \texttt{bb}\rangle$ has among its solutions the string $abbaabbabbabb = u_1 u_3 u_1 u_1 u_3 u_2 u_2 = v_1 v_3 v_1 v_1 v_3 v_2 v_2$, obtained from the index sequence 1311322. For the sake of clarity, we show here both the instance and the solution in tabular form:

i	1	2	3
u_i	abb	b	a
v_i	a	abb	bb

i	1	3	1	1	3	2	2
u_i	abb	a	abb	abb	a	b	b
v_i	a	bb	a	a	bb	abb	abb

The language of solutions to a given instance is context-sensitive. That is, it is possible for a linear bounded automaton to determine whether a given string s is a solution, simply by considering all index sequences of length $\leq 2|s|$.[6]

What is not decidable, in general, is whether this context-sensitive language of solutions is non-empty. To put this another way, the set of PCP instances with at least one solution is not recursive (although it is recursively enumerable).

An instance of PCP can be represented as a 2-tape automaton, $A^{(2)}$, with a unique state, that is both initial and final, and p transitions labeled with pairs of strings $u_i : v_i$, as illustrated in Figure 6a. The set of all solutions to this instance equals $\pi_{\langle 1\rangle}(\sigma_{\{1=2\}}(\mathcal{R}(A^{(2)})))$. If one wishes instead to obtain the language of index sequences of each solution, one can represent the instance as a 3-tape automaton $A^{(3)}$ with an additional tape of indices $i \in [\![1, p]\!]$, as illustrated in Figure 6b, and construct $\pi_{\langle 1\rangle}(\sigma_{\{2=3\}}(\mathcal{R}(A^{(3)})))$.

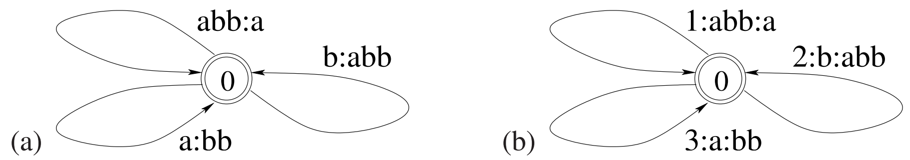

Figure 6: An instance of a PCP (a) without and (b) with an additional tape of indices

This reduction from PCP to auto-intersection demonstrates that it is undecidable whether the result of an unweighted 2-tape auto-intersection is empty.

Furthermore, this implies that there can be no partial auto-intersection algorithm that is "complete" in that it always returns a correct FSM if the auto-intersection is rational, and always terminates with an error code otherwise. If such an algorithm did exist, one could use it as follows to determine the emptiness of an unweighted auto-intersection (and hence to determine the existence of a solution to a PCP instance, which is impossible in general). If the algorithm returned an FSM, we would test it for emptiness by determining whether there was at least one path from an initial to a final state. If the algorithm returned an error code, we would know that the result was non-rational and hence could not be empty.

Despite this gloomy result, some recent work (Zhao, 2002) has explored heuristic tests that can identify some PCP instances as empty, as well as heuristic search methods that try to find a single solution

[6]We may assume without loss of generality that $\langle \varepsilon, \varepsilon\rangle$ is not among the strings in the instance. Then if $s = u_{i_1} u_{i_2} \ldots u_{i_r} = v_{i_1} v_{i_2} \ldots v_{i_r}$ is a solution, we have $r \leq |u_{i_1} u_{i_2} \ldots u_{i_r} v_{i_1} v_{i_2} \ldots v_{i_r}| = |ss| = 2|s|$.

to a PCP quickly (although not the full language of solutions). These methods might provide a starting point for constructing a useful partial algorithm for auto-intersection.

5 Conclusion

We have provided definitions and notation for the central operations on weighted n-ary relations and the finite-state machines that describe the rational cases. Our notation is informed by regarding these objects as weighted databases. This perspective is pedagogically useful and motivates potential applications.

We focused primarily on the important join operation $\bowtie$, and the related operations of generalized composition $\boxtimes$ and auto-intersection $\sigma_{\{1=2\}}$. In some cases, these operators preserve rationality. In general, they do not, and we showed that the resulting relations, while individually decidable (at the level of individual tuples), can have undecidable emptiness as a class.

Our question for future research is whether there exists a partial or approximate algorithm for auto-intersection that can handle some practical cases of infinite relations. This would imply the existence of a similar algorithm for join.

There is some precedent for such investigations. Regarding partial algorithms, we already noted the work of (Zhao, 2002) on partial solutions to the generally undecidable Post's Correspondence Problem, which reduces to our problem. In the speech and language processing community, researchers manage to make practical use of a WFSM determinization algorithm that is not guaranteed to terminate when no answer exists (Mohri, 1997).[7] As for approximations, context-free languages are not in general rational, but they can be usefully approximated by FSMs that accept a close superset or subset (Nederhof, 2000). Approximation by pruning is an option for FSMs weighted by probabilities. Where a naive auto-intersection algorithm would run forever, in an attempt to generate an infinite-state machine, it might be possible to obtain a reasonable finite-state machine by pruning away work on low-probability paths.

Acknowledgments

We wish to thank Mark-Jan Nederhof for discussion of our work at an earlier stage. It was he who saw the relationship between auto-intersection and Post's correspondence problem.

References

Bangalore, Srinivas and Michael Johnston. 2000. Finite-state multimodal parsing and understanding. In *Proc. of the 17th COLING*, pages 369–375, Saarbrücken, Germany, August.

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

Eisner, Jason. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, July.

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.

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

[7] Even though Mohri also exhibits a terminating algorithm for that problem.

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

Kempe, André. 2004. NLP applications based on weighted multi-tape automata. In *Proc. 11th Conf. TALN*, pages 253–258, Fes, Morocco, April.

Kempe, André, Franck Guingne, and Florent Nicart. 2004. Algorithms for weighted multi-tape automata. Research report 2004/031, Xerox Research Centre Europe, Meylan, France. (available from www.xrce.xerox.com and www.arXiv.org/abs/cs.CL/0406003).

Kiraz, George Anton. 2000. Multitiered nonlinear morphology using multitape finite automata: a case study on Syriac and Arabic. *Computational Lingistics*, 26(1):77–105, March.

Knight, Kevin and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4).

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

Lehmann, Daniel J. 1977. Algebraic structures for transitive closure. *Theoretical Computer Science*, 4(1):59–76.

Livescu, Karen, James Glass, and Jeff Bilmes. 2003. Hidden feature models for speech recognition using dynamic Bayesian networks. In *8th European Conference on Speech Communication and Technology (Eurospeech)*.

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

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.

Nederhof, Mark-Jan. 2000. Practical experiments with regular approximation of context-free languages. *Computational Linguistics*, 26(1).

Post, Emil. 1946. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52:264–268.

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

Rosenberg, Arnold L. 1964. On n-tape finite state acceptors. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 76–81.

Sakarovitch, Jacques. 2003. *Éléments de theorie des automates*. Éditions Vuibert, Paris, France.

Zhao, Ling. 2002. Tackling Post's Correspondence Problem. In Jonathan Schaeffer, Martin Müller, and Yngvi Björnsson, editors, *Proceedings of the Third International Conference on Computers and Games, CG 2002*, volume 2883 of *Lecture Notes in Computer Science*, pages 326–344, Edmonton, Canada, July 25-27. Springer-Verlag. Revised edition (January 1, 2004).