EFFICIENT ALGORITHMS FOR TESTING THE TWINS PROPERTY

CYRIL ALLAUZEN

AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07932, USA
e-mail: `allauzen@research.att.com`

and

MEHRYAR MOHRI

AT&T Labs – Research
180 Park Avenue
Florham Park, NJ 07932, USA
e-mail: `mohri@research.att.com`

ABSTRACT

Weighted automata and transducers are powerful devices used in many large-scale applications. The efficiency of these applications is substantially increased when the automata or transducers used are deterministic. There exists a general determinization algorithm for weighted automata and transducers that is an extension of the classical subset construction used in the case of unweighted finite automata [14]. However, not all finite-state transducers or weighted automata and transducers can be determinized using that algorithm, thus the question of the determinizability in that sense is essential. There exists a characterization of the determinizability of functional finite-state transducers and that of unambiguous weighted automata over the tropical semiring based on a general *twins property*. In the case of finite-state transducers, we give an efficient algorithm for testing functionality in time $O(|Q|^2 \, |\Delta| + |E|^2)$ where Q is the set of states, E the set of transitions, and Δ the ouput alphabet of the input transducer. We also present a new and computationally more efficient algorithm for testing the twins property whose complexity is $O(|Q|^2(|Q|^2 + |E|^2))$. In the automata case, we present a new and substantially more efficient algorithm for testing the twins property for unambiguous and cycle-unambiguous weighted automata over commutative and cancellative semirings whose complexity is $O(|Q|^2 + |E|^2)$, which we conjecture to be optimal. Our experiments show our algorithms for testing the twins property to be practical with large weighted automata and transducers of several million transitions found in speech recognition applications.

Keywords: Finite-state transducers, weighted automata, rational power series, determinization, twins property, algorithms.

1. Introduction

Finite automata are classical computational devices used in a variety of large-scale applications [1]. Some applications such as text, speech or image processing require more general devices, *weighted automata*, to account for the variability of the data and to rank alternative hypotheses [14, 9]. A weighted automaton is a finite automaton in which each transition carries some weight in addition to the usual symbol. Finite-state transducers are automata whose transitions are additionally labeled with an output label.

Weighted automata and transducers provide a common representation for the components of complex systems in many applications. These components can be combined efficiently using general algorithms such as composition of weighted transducers [16]. The time efficiency of such systems is substantially increased when *deterministic* or *subsequential* machines are used [14] and the size of these machines can be further reduced using general minimization algorithms [14, 15]. A weighted automaton or transducer is *deterministic* or *subsequential* if it has a unique initial state and if no two transitions leaving the same state share the same input label.

A general determinization algorithm for weighted automata and transducers was introduced by [14]. The algorithm is an extension of the classical subset construction used for unweighted finite automata and outputs a deterministic machine equivalent to the input weighted automaton or transducer. But, unlike the case of unweighted automata, not all finite-state transducers or weighted automata and transducers can be determinized using this algorithm. In fact, some machines do not even admit any equivalent subsequential one, they are not *subsequentiable*. Thus, it is important to design an algorithm for testing the determinizability of finite-state transducers and weighted automata.

A characterization of subsequentiable finite-state transducers based on a *twins property* was given by [6, 7]. The twins property was also shown to be decidable by the same author [4]. The first polynomial-time algorithm for deciding the twins property for functional transducers was given by [20], its time complexity is $O(|Q|^4(|Q|^2 + |E|^2)|\Delta|)$ where Q is the set of states of the input transducer, E the set of its transitions and Δ the output alphabet.[1] More recently, [3] proposed a similar polynomial-time algorithm for deciding the twins property for functional transducers whose complexity is $O(|Q|^4(|Q|^2 + |E|^2))$.[2] This complexity only differs from that of [20] by the fact that it does not depend on the size of the output alphabet.

We present a new and computationally more efficient algorithm for testing the twins property for finite-state transducers. The worst case complexity of our algorithm is $O(|Q|^2(|Q|^2 + |E|^2))$. It is based on a general algorithm of composition of finite-state transducers and a new characterization of the twins property in terms of combinatorics of words.

In the case of weighted automata, a similar twins property was introduced by

---

[1] This is based on our most favorable estimate of the complexity of that algorithm, the authors do not give a precise analysis of the complexity of their algorithm.

[2] This is the correct and updated expression of the complexity as provided by the journal version of the paper [3].

[14] to characterize the determinizability of unambiguous weighted automata over the tropical semiring. The property was also shown to be a sufficient condition for the determinizability of ambiguous machines. [14] also gave an algorithm for testing the twins property for unambiguous weighted automata over a commutative semiring with cancellative multiplication. The complexity of an implementation of this algorithm was analyzed by [5] to be $O(|E|^2|Q|^6)$. We present a new and substantially more efficient algorithm for testing the twins property for unambiguous and cycle-unambiguous weighted automata over commutative and cancellative semirings whose complexity is $O(|Q|^2 + |E|^2)$. We further conjecture this complexity to be optimal.

The first step of our algorithms, in both the transducer and weighted automata cases, consists of constructing the intersection or composition of the *inverse* of the input machine M and itself, $M^{-1} \circ M$, which can be done in quadratic time $O(|Q|^2 + |E|^2)$. The cost of the remaining work in the automata case is linear in the size of the result of composition.

We also give an efficient algorithm to test the *functionality* of a transducer T, that is to determine if T represents a function, using the composed machine $T^{-1} \circ T$. The complexity of our algorithm is $O(|Q|^2 |\Delta| + |E|^2)$ where Δ is the output alphabet.

Our algorithms can be used to test the twins property for weighted transducers, which are the representations commonly used in speech recognition applications. We report experimental results demonstrating their practicality in such applications.

The paper is organized as follows. Section (2) introduces the definitions and notation used in the following sections and briefly describes the composition algorithm for weighted automata and transducers which is used in the first step of several of our algorithms. Section (3) gives a brief overview of a general determinization algorithm, introduces the twins property, and presents a number of characterization results for various classes of semirings. In Section (4), we describe our algorithm for testing the twins property for weighted automata. Section (5) gives some basic definitions and presents several combinatorial results related to string *residues* that are crucial for the proofs and the design of the algorithms introduced in the next sections. Section (6) presents our algorithm for testing the functionality of a finite-state transducer, and Section (7) our algorithm for testing the twins property for finite-state transducers. Finally, Section (8) briefly reports experimental results showing that our algorithms are practical even with the large automata and transducers found in large-vocabulary speech recognition applications.

2. Preliminaries

2.1. Definitions

Weighted automata are automata in which the transitions are labeled with weights in addition to the usual alphabet symbols. For various operations to be well-defined, the weight set needs to have the algebraic structure of a semiring [13] or a left semiring [15].

Definition 1 *A* right semiring *(*left semiring*) is a system* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *where:*

- $(\mathbb{K}, \oplus, \overline{0})$ *is a commutative monoid with* $\overline{0}$ *as the identity element for* $\oplus$*,*

- $(\mathbb{K}, \otimes, \overline{1})$ *is a monoid with* $\overline{1}$ *as the identity element for* $\otimes$*,*

- $\otimes$ *right (resp. left) distributes over* $\oplus$*:* $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$ *(resp.* $c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)$*),*

- $\overline{0}$ *is an annihilator for* $\otimes$*:* $\forall a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$*.*

$(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ *is a* semiring *if it is both a left and a right semiring.*

A semiring is *commutative* when $\otimes$ is commutative. Thus, a semiring is a ring that may lack negation. Some classical examples of semirings are the tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ or the semiring of non-negative integers $(\mathbb{N}, +, \times, 0, 1)$. The *string semiring* $(\Sigma^* \cup \infty, \wedge, \cdot, \infty, \epsilon)$, where $\wedge$ denotes the longest common prefix operation, $\cdot$ concatenation and ∞ an additional element (such that, for all string w, $w \wedge \infty = \infty \wedge w = w$ and $w \cdot \infty = \infty \cdot w = \infty$), is a left semiring [15]. The multiplicative operation of a semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is said to be *cancellative* if for any x, x' and x_0 in $\mathbb{K}$ such that $x_0 \neq \overline{0}$, $x \otimes x_0 = x' \otimes x_0$ implies $x = x'$. The semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is then also said to be cancellative.

A semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is said to be *left divisible* if for any $x \neq \overline{0}$, there exists $y \in \mathbb{K}$ such that $y \otimes x = \overline{1}$, that is if all elements of $\mathbb{K}$ admit a left inverse. $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is said to be *weakly left divisible* if for any x and y in $\mathbb{K}$ such that $x \oplus y \neq \overline{0}$, there exists at least one z such that $x = (x \oplus y) \otimes z$. When the $\otimes$ operation is cancellative, z is unique and we can then write: $z = (x \oplus y)^{-1} x$. When z is not unique, we can still assume that we have an algorithm to find one of the possible z and call it $(x \oplus y)^{-1} x$. Furthermore, we will assume that z can be found in a consistent way, that is: $((u \otimes x) \oplus (u \otimes y))^{-1}(u \otimes x) = (x \oplus y)^{-1} x$ for any $x, y, u \in \mathbb{K}$ such that $u \neq \overline{0}$. A semiring is *zero-sum-free* if for any x and y in $\mathbb{K}$, $x \oplus y = \overline{0}$ implies $x = y = \overline{0}$. In the following definitions, $\mathbb{K}$ is assumed to be a left semiring or a semiring.

Definition 2 *A* weighted automaton $A = (\Sigma, Q, I, F, E, \lambda, \rho)$ *over* $\mathbb{K}$ *is a 7-tuple where:*

- Σ *is the finite alphabet of the automaton,*

- Q *is a finite set of states,*

- $I \subseteq Q$ *the set of initial states,*

- $F \subseteq Q$ *the set of final states,*

- $E \subseteq Q \times \Sigma \times \mathbb{K} \times Q$ *a finite set of transitions,*

- $\lambda : I \to \mathbb{K}$ *the initial weight function mapping* I *to* $\mathbb{K}$*, and*

- $\rho : F \to \mathbb{K}$ *the final weight function mapping* F *to* $\mathbb{K}$*.*

Definition 3 *A* finite-state transducer $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$ *is an 8-tuple where:*

- Σ *is the finite input alphabet of the transducer,*

- Δ *is the finite output alphabet,*

- Q *is a finite set of states,*

- $I \subseteq Q$ *the set of initial states,*

- $F \subseteq Q$ *the set of final states,*

- $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times Q$ *a finite set of transitions,*

- $\lambda : I \to \Delta^*$ *the initial output function mapping I to Δ^*, and*

- $\rho : F \to \Delta^*$ *the final output function mapping F to Δ^*.*

Given a transition $e \in E$, we denote by $i[e]$ its input label, $p[e]$ its origin or previous state and $n[e]$ its destination state or next state, $w[e]$ its weight (weighted automata case), o[e] its output label (transducer case). Given a state $q \in Q$, we denote by $E[q]$ the set of transitions leaving q.

A *path* $\pi = e_1 \cdots e_k$ in A is an element of E^* with consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. We extend n and p to paths by setting: $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$. We denote by $P(q, q')$ the set of paths from q to q' and by $P(q, x, q')$ the set of paths from q to q' with input label $x \in \Sigma^*$. These definitions can be extended to subsets $R, R' \subseteq Q$, by: $P(R, x, R') = \bigcup_{q \in R, \, q' \in R'} P(q, x, q')$. The labeling functions i (and similarly o) and the weight function w can also be extended to paths by defining the label of a path as the concatenation of the labels of its constituent transitions, and the weight of a path as the $\otimes$-product of the weights of its constituent transitions: $i[\pi] = i[e_1] \cdots i[e_k]$, $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. We also extend w to any finite set of paths Π by setting: $w[\Pi] = \bigoplus_{\pi \in \Pi} w[\pi]$. The output weight associated by an automaton A to an input string $x \in \Sigma^*$ is defined by:

$$[\![A]\!](x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$$

$[\![A]\!](x)$ is defined to be $\overline{0}$ when $P(I, x, F) = \emptyset$. The set of output strings associated by a transducer T to an input string x is defined by:

$$[\![T]\!](x) = \bigcup_{\pi \in P(I,x,F)} \lambda(p[\pi]) \cdot o[\pi] \cdot \rho(n[\pi])$$

$[\![T]\!](x) = \emptyset$ when $P(I, x, F) = \emptyset$. The domain of definition of T is defined as: $Dom(T) = \{x \in \Sigma^* : [\![T]\!](x) \neq \emptyset\}$. A transducer is *functional* or *single-valued* if it associates at most one string to any input string x, that is if $|[\![T]\!](x)| \leq 1$. Functional transducers can be viewed as weighted automata over the string semiring [14].

A *successful path* in a weighted automaton or transducer M is a path from an initial state to a final state. A state q of M is *accessible* if q can be reached from I. It is *coaccessible* if a final state can be reached from q. A weighted automaton M is *trim* if there is no transition with weight $\overline{0}$ in M and if all states of M are both accessible and coaccessible. M is *cycle-unambiguous* if for any state q and any string x there is

at most one cycle in q labeled with x. M is *unambiguous* if for any string $x \in \Sigma^*$ there is at most one successful path labeled with x. Thus, an unambiguous transducer is functional.

2.2. Intersection and Composition

Intersection of weighted automata and composition of finite-state transducers are both special cases of composition of *weighted transducers*, that is transducers whose transitions are labeled with some weight in addition to the input and output symbols. The semiring need to be commutative for this construction to work. States in the composition $T_1 \circ T_2$ of two weighted transducers T_1 and T_2 are identified with pairs of a state of T_1 and a state of T_2.[3] Leaving aside transitions with ϵ inputs or outputs, the following rule specifies how to compute a transition of $T_1 \circ T_2$ from appropriate transitions of T_1 and T_2:[4]

$$(q_1, a, b, w_1, q_2) \quad \text{and} \quad (q'_1, b, c, w_2, q'_2) \Longrightarrow ((q_1, q_2), a, c, w_1 \otimes w_2, (q'_1, q'_2))$$

Intersection corresponds to the case where input and output labels of transitions are identical and the composition of unweighted transducers is obtained by simply omitting the weights. Thus, we can use both the notation $A = A_1 \cap A_2$ or $A_1 \circ A_2$ for the intersection of two weighted automata A_1 and A_2. A string x is recognized by A iff it is recognized by both A_1 and A_2 and $[\![A]\!](x) = [\![A_1]\!](x) \otimes [\![A_2]\!](x)$.

Similarly, given two strings $x \in \Sigma^*$ and $y \in \Theta^*$, $y \in [\![T_1 \circ T_2]\!](x)$ iff there is a string $z \in \Delta^*$ such that $z \in [\![T_1]\!](x)$ and $y \in [\![T_2]\!](z)$. The *inverse of* T is the transducer denoted by T^{-1} and obtained from T by transposing the input and output labels of each transition of T. Note that, given two string $x \in \Sigma^*$ and $y \in \Delta^*$, $x \in [\![T^{-1}]\!](y)$ iff $y \in [\![T]\!](x)$. In the following, we will assume without loss of generality that the initial weight or output function λ is respectively $\overline{1}$ and ϵ to simplify the presentation.

3. Determinization and the twins property

3.1. Deterministic or subsequential machines

A weighted automaton or a finite-state transducer M is *deterministic* or *subsequential* if it has a deterministic input [19], that is if it has a unique initial state and if no two transitions leaving the same state share the same input label. A weighted automaton or a finite-state transducer M is *subsequentiable* if there exists a subsequential machine M' equivalent to M: $[\![M]\!] = [\![M']\!]$.

In what follows, we assume that the weighted automata considered are all such that for any string $x \in \Sigma^*$, $w[P(I, x, Q)] \neq \overline{0}$. This condition is always satisfied with trim machines over the tropical semiring, the string semiring or any zero-sum-free semiring.

---

[3]We use a *matrix notation* for the definition of composition as opposed to a *functional notation*.

[4]See [16] for a detailed presentation of the algorithm including the use of a filter for dealing with ϵ-multiplicity in the case of non-idempotent semirings
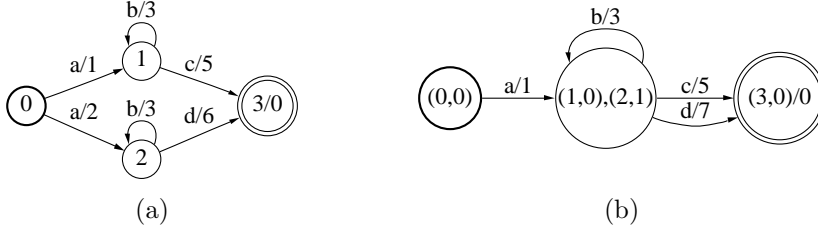
Figure 1: Determinization of a weighted automaton over the tropical semiring.
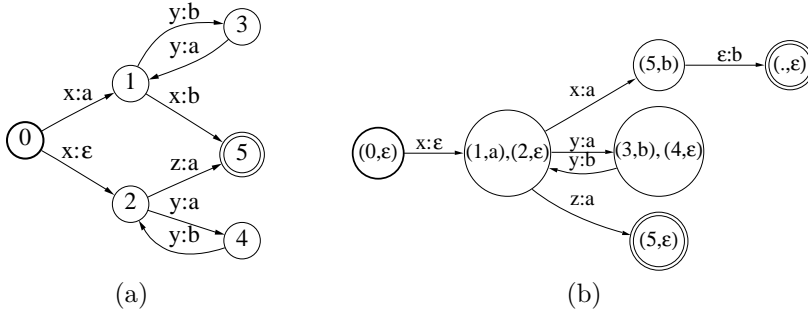

Figure 2: Determinization of a functional transducer.

3.2. Determinization

There exists a natural extension of the classical subset construction to the case of weighted automata over a weakly left divisible left semiring called *determinization* [14]. The algorithm is generic: it works with any weakly left divisible left semiring. It covers in particular the case of finite-state transducers since the string semiring is weakly left divisible.

The states of the result of the determinization of a weighted automaton $A = (Q, I, F, \Sigma, \delta, \sigma, \lambda, \rho)$ correspond to *weighted subsets* $\{(q_0, w_0), \ldots, (q_n, w_n)\}$ where each $q_i \in Q$ is a state of the input machine, and w_i a remainder weight. The algorithm starts with the subset reduced to $\{(i, \lambda(i)) : i \in I\}$ and proceeds by creating a transition labeled with $a \in \Sigma$ and weight w leaving $\{(q_0, w_0), \ldots, (q_n, w_n)\}$ if there exists at least one state q_i admitting an outgoing transition labeled with a, w being defined by:[5]

$$w = \bigoplus_{e \in E[q_i],\ l[e]=a} w_i \otimes w[e]$$

Fig. (1) illustrates the application of the algorithm to an input weighted automaton over the tropical semiring. A state r of the output automaton that can be reached from the start state by a path π corresponds to the set of pairs $(q, x) \in Q \times \mathbb{K}$ such that q can be reached from an initial state of the original machine by a path σ with label $i[\sigma] = i[\pi]$ and weight $w[\sigma]$ such that $\lambda(p[\sigma]) \otimes w[\sigma] = \lambda(p[\pi]) \otimes w[\pi] \otimes x$. Thus,

---

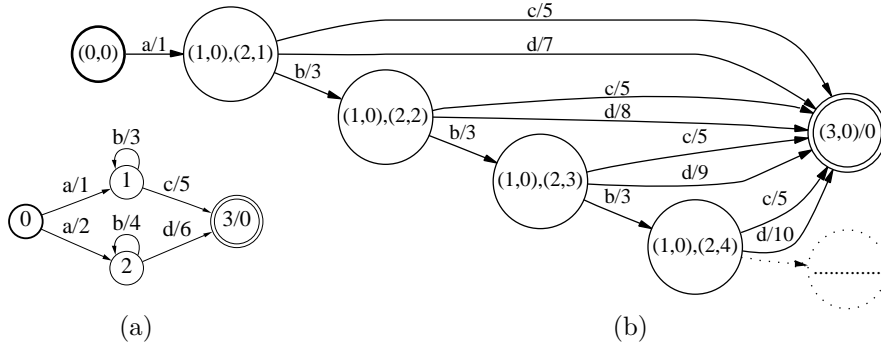[5]See [14] for a full description of the algorithm.

Figure 3: (a) Non-determinizable weighted automaton over the tropical semiring; states 1 and 2 are non-twin siblings. (b) The first states created by determinization applied to the automaton of Figure (a). The algorithm does not halt and produces an infinite number of states in this case.
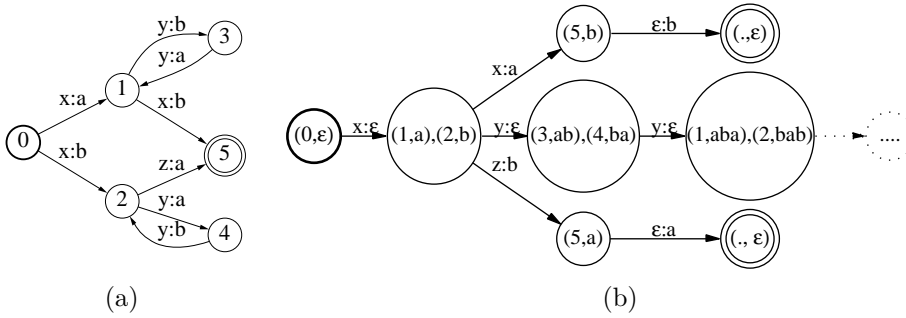

Figure 4: (a) Non-determinizable functional transducer, or weighted automaton over the string semiring; states 1 and 2 are non-twin siblings. (b) Determinization does not halt and creates an infinite number of states in this case.

x can be viewed as the remainder weight at state q.

Similarly, Fig. (2) illustrates the determinization of a functional transducer, or, equivalently, a weighted automaton over the string semiring. Here, the weighted subsets are made of pairs (q, x) where q is a state of the original transducer and x a remainder string.

Unlike the unweighted case, determinization does not halt for some input weighted automata. This is clear since some weighted automata are not even subsequentiable. Fig. (3) and Fig. (4) show a weighted automaton over the tropical semiring and a functional finite-state transducer for which determinization does not halt and that are not subsequentiable. In what follows, we say that a weighted automaton or a finite-state transducer M is *determinizable* if the determinization algorithm of [14] halts for the input M. With a determinizable input, the algorithm outputs an equivalent subsequential weighted automaton. Note that any acyclic weighted automaton is determinizable because the number of weighted subsets created by the algorithm is at most equal to the number of distinct strings labeling the paths of the original machine
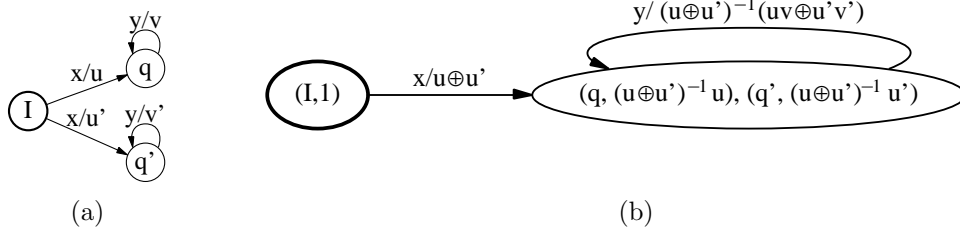
Figure 5: Illustration of the relationship between the twins property and determinization.

starting at an initial state.

By definition, determinizability implies subsequentiability, but the converse does not always hold: in some semirings, there are subsequentiable weighted automata that are not determinizable. In general, it is not known if the sequentiability or the determinizability of a machine M is decidable. Some results are available for specific semirings or classes of semirings however. In particular, in the string semiring, both subsequentiability and determinizability are known to be decidable and are in fact equivalent for trim machines, see Section (3.5). Partial decidability results are also available for the tropical semiring, see Section (3.4). For both of these semirings, the results and characterizations are based on a property of the input automaton or transducer that we will describe in the next section.

3.3. The twins property

This property was first formulated for finite-state transducers by [7, 4] and for weighted automata over the tropical semiring by [14]. Here, we give a general formulation of this property for weighted automata over any weakly left divisible left semiring not necessary commutative. The definition covers in particular the case of finite-state transducers.

Definition 4 *Let A be a weighted automaton over a weakly left divisible left semiring* $\mathbb{K}$*. Two states q and q' of A are said to be* siblings *if there exist two strings x and y in* Σ^* *such that both q and q' can be reached from I by paths labeled with x and there is a cycle at q and a cycle at q' both labeled with y. Two sibling states q and q' are said to be* twins *if for any strings x and y, the following equation and its symmetric counterpart obtained by transposing q and q' hold:*

$$(w[P(I, x, q)] \oplus w[P(I, x, q')])^{-1} w[P(I, x, q')] =$$
$$((w[P(I, x, q)] \otimes w[P(q, y, q)]) \oplus (w[P(I, x, q')] \otimes w[P(q', y, q')]))^{-1}$$
$$w[P(I, x, q')] \otimes w[P(q', y, q')] \quad (1)$$

A has the twins property *if any two sibling states of A are twins.*

Figs. (5)(a)-(b) illustrate the connection between the twins property and determinization. States q and q' of the automaton of Fig. (5) (a) are siblings. Determinization

creates a state r corresponding to the subset $\{(q, (u \oplus u')^{-1}u), (q', (u \oplus u')^{-1}u')\}$. For determinization to terminate and not to create an infinite number of states, there needs to be a cycle with input label y at state r. For this cycle to exists, the subset corresponding to the destination state and the one corresponding to r must coincide, in particular the weight associated to q must be the same in both subsets:

$$[(u \oplus u')^{-1}((u \otimes v) \oplus (u' \otimes v'))]^{-1}((u \oplus u')^{-1}u \otimes v) = (u \oplus u')^{-1}u$$

which can be rewritten as: $((u \otimes v) \oplus (u' \otimes v'))^{-1}(u \otimes v) = (u \oplus u')^{-1}u$, that is the twins property.

In the following paragraphs, we will present the decidability results known for several classes of semirings and several characterizations based on the twins property.

3.4. Commutative and cancellative semirings

In this section, we assume that the semiring $\mathbb{K}$ is commutative and that its multiplicative operation is cancellative.

3.4.1. Simpler formulation of the twins property

Proposition 1 *Assume that $\mathbb{K}$ is commutative and cancellative, then two sibling states q and q' are twins iff for any string y:*

$$w[P(q, y, q)] = w[P(q', y, q')] \tag{2}$$

Proof. Let q and q' be two sibling states. For any strings x and y, Eq. (1) can then be rewritten as:
$$(u \oplus u')^{-1}u = (uv \oplus u'v')^{-1}uv$$

if we let $u = w[P(I, x, q)]$, $u' = w[P(I, x, q')]$, $v = w[P(q, y, q)]$ and $v' = w[P(q', y, q')]$. Multiplying both sides of this equality by $(u \oplus u')(uv \oplus u'v')$ gives:

$$(u \oplus u')(uv \oplus u'v')(u \oplus u')^{-1}u = (u \oplus u')(uv \oplus u'v')(uv \oplus u'v')^{-1}uv$$

Since the semiring is commutative, this can be reduced to $(uv \oplus u'v')u = (u \oplus u')uv$. Since multiplication is cancellative, factor u can be removed from both sides:

$$(uv \oplus u'v') = (u \oplus u')v \tag{3}$$

Similarly, the symmetric version of Eq. (1) for q' and q leads to:

$$(uv \oplus u'v') = (u \oplus u')v' \tag{4}$$

Eqs. (3) and (4) imply that $(u \oplus u')v = (u \oplus u')v'$, which gives $v = v'$ since $\otimes$ is cancellative. Conversely, assume that $v = v'$, then:

$$(uv \oplus u'v')^{-1}uv = (uv \oplus u'v)^{-1}uv = v^{-1}(u \oplus u')^{-1}uv$$

Using the commutativity of $\otimes$, we obtain: $(uv \oplus u'v')^{-1}uv = (u \oplus u')^{-1}u$. $\qquad\square$
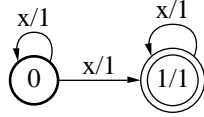
Figure 6: Infinitely ambiguous weighted automaton over the real semiring which has the twins property but is not determinizable.

3.4.2. Tropical semiring

Eq. (2) corresponds to our original formulation of the twins property for weighted automata [14]. We had proved that the twins property is a sufficient condition for determinizability in the tropical semiring using that formulation.

Theorem 1 ([14]) *Let A be a weighted automaton over the tropical semiring. If A has the twins property, then A is determinizable.*

With trim unambiguous weighted automata, the condition is also necessary.

Theorem 2 ([14]) *Let A be a trim unambiguous weighted automaton over the tropical semiring. Then the three following properties are equivalent:*

1. *A is determinizable.*
2. *A has the twins property.*
3. *A is subsequentiable.*

The theorem motivates the need for an efficient algorithm testing the twins property for weighted automata over the tropical semiring.

3.4.3. Other commutative and cancellative semirings

Theorem 2 can be generalized to the real semiring and the log semirings using the proofs given by [14]. But, the equivalent of theorem 1 does not hold for these semirings. Some infinitely ambiguous weighted automata over the real or log semirings are not subsequentiable and thus not determinizable despite they have the twins property. Fig. (6) shows a simple example of such a weighted automaton.

3.5. The string semiring

Eq. (1) of Section (3.3) defining the twins property can be simplified in the particular case of the string semiring and re-written as: for any paths $\pi_1 \in P(I, y, q)$, $\pi_2 \in P(q, x, q)$, $\pi'_1 \in P(I, y, q')$, $\pi'_2 \in P(q', x, q')$,

$$o[\pi_1]^{-1}o[\pi'_1] = (o[\pi_1]o[\pi_2])^{-1}o[\pi'_1]o[\pi'_2] \tag{5}$$

if we adopt the notation of the first definition of finite-state transducers given in Section (2.1). The twins property provides a characterization of the determinizability of functional transducers.

Theorem 3 *Let T be a trim functional transducer. Then the three following properties are equivalent:*

1. *T is determinizable.*

2. *T has the twins property.*

3. *T is subsequentiable.*

The equivalence of the last two properties was shown by [7] who also showed that the twins property is decidable. The proof of the equivalence of the first two statements is a special case of a general proof given by [2]. Since the twins property is decidable, the theorem also implies that subsequentiability and determinizability are decidable for trim functional transducers. We will present efficient algorithms for testing functionality and the twins property for finite-state transducers.

Determinization can be extended to output *finitely subsequential transducers*, that is deterministic transducers with a finite number of final outputs. With that extension, theorem 3 can be generalized by relaxing the condition on the functionality of the transducer, and by replacing subsequentiable transducers by *finitely subsequentiable transducers*, that is transducers that admit an equivalent finitely subsequential transducer.

Theorem 4 ([2]) *Let T be a trim finite-state transducer. Then the following three properties are equivalent:*

1. *T is determinizable.*

2. *T has the twins property.*

3. *T is finitely subsequentiable.*

3.6. Finite semirings

The case of finite semirings is trivial since only a finite number of distinct weighted subsets can be constructed by determinization in such semirings. Thus all weighted automata over a finite semiring are determinizable and subsequentiable. The specific case of the Boolean semiring or unweighted automata is a classical result of automata theory.

4. Test of the twins property for weighted automata

In this section, we assume that the semiring $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is commutative and cancellative. By proposition 1, two sibling states q and q' of a weighted automaton over $\mathbb{K}$ are twins iff for any string y: $w[P(q, y, q)] = w[P(q', y, q')]$.

Since the multiplicative operation of the semiring $\mathbb{K}$ is cancellative, an inverse can be simulated externally by considering the semiring $\mathbb{K}' = (\mathbb{K} \times \mathbb{K})/\equiv$ where $\equiv$ denotes the equivalence relation defined by $(x, y) \equiv (z, t)$ iff $x \otimes t = y \otimes z$. $\mathbb{K}$ can then be embedded into $\mathbb{K}'$, indeed each $x \in \mathbb{K}$ can then be identified with $(x, \overline{1})$ and admits
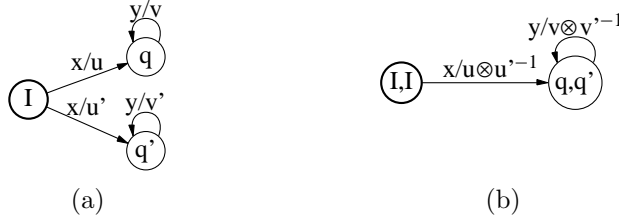
Figure 7: (a) Weighted automaton A with sibling states q and q'. (b) Intersection machine $A \cap A^{-1}$. The paths from I to q and I to q' with the same labels are matched by composition. By proposition 1, q and q' are twins iff $v = v'$, that is iff the weight $v \otimes v'^{-1}$ of the cycle in $A \cap A^{-1}$ is $\overline{1}$.

$(\overline{1}, x)$ as an inverse. Thus, we can freely assume that $\mathbb{K}$ admits an inverse in the rest of the section.

Our algorithm for testing the twins property is based on the intersection of weighted automata. We define A^{-1} as the machine obtained from A by replacing each non-$\overline{0}$ weight by its inverse.

Theorem 5 *Let A be a trim cycle-unambiguous weighted automaton over $\mathbb{K}$. A has the twins property iff the weight of any cycle in $A \cap A^{-1}$ is $\overline{1}$.*

Proof. Let A be a trim cycle-unambiguous weighted automaton over $\mathbb{K}$. Since A is trim, the transition weights of A are all non-$\overline{0}$. By definition of the weighted intersection algorithm, two states q_1 and q_2 of A are siblings iff the state $R = (q_1, q_2)$ is constructed by the intersection algorithm and there is a cycle c at R. Let q_1 and q_2 be two such states, and let x be the label of c. The weight of c is the $\otimes$-product of the weights of a cycle c_1 in A at state q_1 labeled with x and the inverse of the weight of a cycle c_2 at q_2 labeled with x: $w[c] = w[c_1] \otimes w[c_2]^{-1}$. Since A is cycle-unambiguous, the cycles c_1 and c_2 are unique. Thus A has the twins property iff $w[c_1] = w[c_2]$, that is iff $w[c] = \overline{1}$. Fig. (7) illustrates the proof of the theorem. $\square$

Example The cycles of the intersected automaton $A \cap A^{-1}$ of Fig. (8)(a) are all weighted with $0 = \overline{1}$. This can be used to verify that the automaton A of Fig. (1)(a) has the twins property. The intersected automaton $A \cap A^{-1}$ of Fig. (8)(b) admits two cycles with non-zero weights: $1 \neq 0$ and $-1 \neq 0$. This confirms that the automaton A of Fig. (3)(a) does not have the twins property.

Checking that the weight of each cycle of a weighted automaton A equals $\overline{1}$ can be done in linear time. Indeed, let S be a strongly connected component of A and q_S an arbitrary state of S. We can run a depth-first search (DFS) of S starting from q_S to compute the weight of any path from q_S to each state $q \in S$. Clearly, that weight must be unique, otherwise there would be two cycles through q_S and q with distinct weights and the weight of one at least would be different from $\overline{1}$. Fig. (9) illustrates this fact. Thus, we can denote by $W[q]$ the weight of a path from q_S to q. We initialize the value of $W[q]$, for $q \in S - \{q_S\}$, by some undefined value UNDEFINED
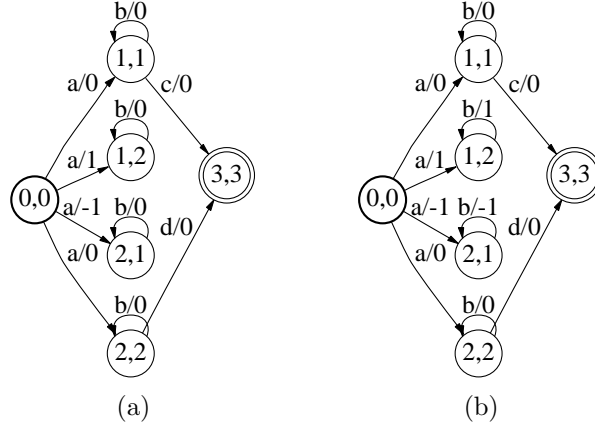
Figure 8: Intersected automata $A \cap A^{-1}$, A being the weighted automaton over the topical semiring of: (a) Fig. (1)(a); (b) Fig. (3)(a).
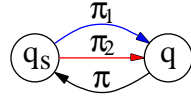

Figure 9: q_S and q are two states of the same strongly connected component of $A \cap A^{-1}$. If there are two paths π_1 and π_2 from q_S to q, then, since q_S and q are in the same strongly connected component, there is also a path π from q to q_S. $\pi_1\pi$ and $\pi_2\pi$ are cycles, thus $w[\pi_1\pi] = w[\pi_2\pi] = \overline{1}$ and $w[\pi_1] = w[\pi_2]$.

and set: $W[q_S] \leftarrow \overline{1}$. The following is the pseudocode of our algorithm.

Cycle_Identity(S)
```
 1   for each e ∈ E    ▷ transitions visited in the order of a DFS of S from q_S
 2       do  if (W[n[e]] = UNDEFINED)
 3               then  W[n[e]] ← W[p[e]] ⊗ w[e]
 4           if (W[n[e]] ≠ W[p[e]] ⊗ w[e])
 5               then  return FALSE
 6   return TRUE
```

Lines 2-3 define $W[n[e]]$ as the weight of the first path from q_S to $n[e]$ found in a DFS of S. Lines 4-5 check that the weight of any other path from q_S to q found in a DFS of S equals $W[n[e]]$ and otherwise output FALSE. If this condition is never violated, the algorithm returns TRUE (line 6).

Cycle_Identity(S) returns TRUE iff the weight of any cycle in S equals $\overline{1}$. Indeed, it returns TRUE iff for all transitions e in S, $W[n[e]] = W[p[e]] \otimes w[e]$. By induction on the length of π, this is equivalent to: $W[n[\pi]] = W[p[\pi]] \otimes w[\pi]$ for any path π in S. If π is a cycle, then $p[\pi] = n[\pi]$, and thus $w[\pi] = \overline{1}$. Conversely, as already pointed out above, if the weight of each cycle equals $\overline{1}$, all paths from q_S to any state q must

have the same weight, in particular those found in a DFS of S and thus the algorithm returns TRUE.

Theorem 6 *There exists an algorithm to test the twins property for any trim cycle-unambiguous weighted automaton A over $\mathbb{K}$ in time $O(|Q|^2 + |E|^2)$.*

Proof. By Theorem 5, the test is equivalent to testing that the weight of any cycle of $B = A \cap A^{-1}$ equals $\overline{1}$. This can be done by running Cycle_Identity for each strongly connected component S of B. The total cost of the algorithm is linear in the size of B since a DFS can be done in linear time and since the strongly connected components of B can also be computed in linear time [8]. Thus the complexity of the algorithm is $O(|Q|^2 + |E|^2)$. □

We conjecture this complexity to be optimal when the alphabet Σ contains at least two distinct elements. Indeed, determining sibling states seems to require matching paths with the same input label and thus computing the intersection machine.

5. Residues

Our test of functionality and our test of the twins property are based on the properties of the *residue* of two strings or that of two paths. Thus, we start with some definitions and give results that are relevant to the proofs presented in the following sections.

We first introduce some classical notions of combinatorics on words, see [17] for a recent survey of the results in this field. Given two strings x and y in Σ^*, we say that y is a *prefix* (*suffix*) of x if there exists $z \in \Sigma^*$ such that $x = yz$ (resp. $x = zy$). Let $\Sigma^{(*)}$ denote the *free group generated by* Σ. Given two elements x and y of $\Sigma^{(*)}$, the *residue of x by y* is defined as: $y^{-1}x$. A residue is said to be *pure* if it is in $\Sigma^* \cup (\Sigma^{-1})^*$. Note that when x and y are strings, $y^{-1}x$ is pure if y is a prefix of x or x is a prefix of y. An element $x \in \Sigma^{(*)}$ is said to be *primitive* if it cannot be written as: $x = y^n$, with $n \in \mathbb{N}$ and $y \neq x$. For any element $x \in \Sigma^{(*)}$, there exists a unique primitive element $y \in \Sigma^{(*)}$ such that $x = y^n$, $n \in \mathbb{N}$, y is called the *primitive root of* x. The following is an essential combinatorial property of the elements of $\Sigma^{(*)}$ that applies in particular to residues.

Lemma 1 ([17]) *Let x and y be two elements of $\Sigma^{(*)} - \{\epsilon\}$. x and y commute, i.e., $xy = yx$, iff x and y, or x and y^{-1}, have the same primitive roots.*

When x and y commute, we write $x \equiv y$. Note that for any string $u \in \Sigma^*$, $x \equiv y$ iff $u^{-1}xu \equiv u^{-1}yu$. The following lemma gives some general properties of commutativity in the free group.

Lemma 2 *Let x, y and z be in $\Sigma^{(*)}$. Then the following properties hold:*

(i) If $x \equiv y$, $x \equiv z$ and $x \neq \epsilon$ then $y \equiv z$ and we can write $x \equiv y \equiv z$.

(ii) If $x \equiv y$ and $x \equiv z$, then $x \equiv yz$.

(iii) If $x \equiv y$, then $x \equiv y^{-1}$.

Proof. The statement (i) follows directly Lemma 1. The proof of (ii) is given by: $xyz = yxz = yzx$ and that of (iii) by: $xb^{-1} = b^{-1}bxb^{-1} = b^{-1}xbb^{-1} = b^{-1}x$. $\qquad\square$

The first statement of the lemma implies the transitivity of the relation $\equiv$ over $\Sigma^{(*)} - \{\epsilon\}$. More generally, $\equiv$ defines an equivalence relation on $\Sigma^{(*)} - \{\epsilon\}$.

Given a path π in a transducer T, we define the *residue of π* as the residue of its input and output labels: $\langle\pi\rangle = i[\pi]^{-1}o[\pi]$. We extend the definition of purity to paths: a path π is *pure* if its residue $\langle\pi\rangle$ is pure. The following properties of path residues will be used in our design of an algorithm for testing the twins property.

Lemma 3 *Let π_1, π_2, π_3 and π be four paths in a transducer T such that $n[\pi_1] = n[\pi_2] = n[\pi_3] = p[\pi]$. Then,*

  (i) $\langle\pi_1\rangle = \langle\pi_2\rangle$ *iff* $\langle\pi_1\pi\rangle = \langle\pi_2\pi\rangle$;

  (ii) $\langle\pi_1\rangle^{-1}\langle\pi_3\rangle \equiv \langle\pi_1\rangle^{-1}\langle\pi_2\rangle$ *iff* $\langle\pi_1\pi\rangle^{-1}\langle\pi_3\pi\rangle \equiv \langle\pi_1\pi\rangle^{-1}\langle\pi_2\pi\rangle$;

  (iii) *if π_1 is not pure, then $\pi_1\pi$ is not pure;*

  (iv) *if π_1 is not pure and $\langle\pi_1\pi\rangle = \langle\pi_1\rangle$, then $i[\pi] = o[\pi] = \epsilon$.*

Proof. The condition $\langle\pi_1\rangle = \langle\pi_2\rangle$ is equivalent to $i[\pi]^{-1}\langle\pi_1\rangle o[\pi] = i[\pi]^{-1}\langle\pi_2\rangle o[\pi]$, which proves (i). The second statement follows the observation that for $x, y, u \in \Sigma^{(*)}$, x and y commute iff $u^{-1}xu$ and $u^{-1}yu$ commute. (iii) and (iv) result from the fact that when π_1 is not pure, there exist two strings x and y with no common non-empty prefix such that $\langle\pi_1\rangle = x^{-1}y$. $\qquad\square$

The following combinatorial result for residues will be used to prove the main theorem of Section 7.

Lemma 4 *Let r, r_1, r_2, s_1 and s_2 be residues in $\Sigma^{(*)}$ such that $s_1^{-1}s_2 \neq \epsilon$. If $r_1^{-1}r_2 \equiv r_1^{-1}s_1$, $r_1^{-1}r_2 \equiv r_1^{-1}s_2$ and $s_1^{-1}s_2 \equiv s_1^{-1}r$, then $r_1^{-1}r_2 \equiv r_1^{-1}r$.*

Proof. By Lemma 2 (iii):

$$r_1^{-1}r_2 \equiv r_1^{-1}s_1 \Longrightarrow r_1^{-1}r_2 \equiv s_1^{-1}r_1$$

By Lemma 2 (ii):

$$(r_1^{-1}r_2 \equiv s_1^{-1}r_1 \text{ and } r_1^{-1}r_2 \equiv r_1^{-1}s_2) \Longrightarrow r_1^{-1}r_2 \equiv (s_1^{-1}r_1)(r_1^{-1}s_2) = s_1^{-1}s_2$$

By hypothesis, $s_1^{-1}s_2 \neq \epsilon$, thus by Lemma 2 (i),

$$(s_1^{-1}s_2 \equiv r_1^{-1}r_2 \text{ and } s_1^{-1}s_2 \equiv s_1^{-1}r) \Longrightarrow r_1^{-1}r_2 \equiv s_1^{-1}r$$

Finally, by Lemma 2 (ii):

$$(r_1^{-1}r_2 \equiv r_1^{-1}s_1 \text{ and } r_1^{-1}r_2 \equiv s_1^{-1}r) \Longrightarrow r_1^{-1}r_2 \equiv (r_1^{-1}s_1)(s_1^{-1}r) = r_1^{-1}r$$

This ends the proof of the lemma. $\qquad\square$

6. Test of functionality for finite-state transducers

The general idea behind the design of our algorithm for testing the functionality of a finite-state transducer T is to test the identity of the finite-state transducer $T^{-1} \circ T$. This idea of reducing functionality to identity was previously suggested in [10]. Functionality was first proved to be decidable for finite-state transducers by [18]. It was later shown to be decidable in polynomial-time by [11]. In [3], the authors give another algorithm for testing the functionality of finite-state transducers whose complexity is $O(|Q|^2(|Q|^2 + |E|^2))$.

6.1. Characterization

We denote by Id_X the restriction of the identity function to a subset $X \subseteq \Sigma^*$. The functionality of a transducer can be tested using the composed machine $T^{-1} \circ T$. For the sake of clarity, we will assume in the following without loss of generality that the transducer T and thus $T^{-1} \circ T$ have a unique initial state.

Lemma 5 *A transducer T is functional iff $[\![T^{-1} \circ T]\!] = Id_{Dom(T^{-1} \circ T)}$.*

Proof. Indeed, by definition of T^{-1}, $y \in [\![T^{-1} \circ T]\!](x)$ iff there exists z such that $y \in [\![T]\!](z)$ and $x \in [\![T]\!](z)$. This proves the lemma. □

6.2. Algorithm

We will show that checking that a transducer T is equivalent to the identity function over its domain of definition can be done in linear time in the size of T. This is equivalent to $i[\pi] = o[\pi]$, or equivalently $\langle \pi \rangle = \epsilon$, for any successful path π. Note that if $[\![T]\!] = Id_{Dom(T)}$, then for any two paths π and π' from the unique initial state s to a coaccessible state q, $\langle \pi \rangle = \langle \pi' \rangle$. Thus, we can denote by $R[q]$ the residue of any path from s to such a state q. The following is the pseudocode of the procedure Identity for checking that $[\![T]\!] = Id_{Dom(T)}$.

Identity(T)
    1 **for** each $e \in E$ such that $coacc[n[e]] = \text{TRUE}$ ▷ in the order of a DFS from s
    2   **do if** $(R[n[e]] = \text{UNDEFINED})$
    3   **then** $R[n[e]] \leftarrow i[e]^{-1} R[p[e]] o[e]$
    4   **if** $(R[n[e]] \neq i[e]^{-1} R[p[e]] o[e])$ **or** $(n[e] \in F$ **and** $R[n[e]] \neq \epsilon)$
    5   **then return** FALSE
    6 **return** TRUE

$R[i]$ is initialized to ϵ and the set of coaccessible states q of T marked by $coacc[q] = \text{TRUE}$. Lines 2-3 define $R[n[e]]$ as the residue of the first path from I to $n[e]$ found in a DFS of T for any coaccessible state $n[e]$ ($coacc[n[e]] = \text{TRUE}$). Lines 4-5 check that the residue of any other path from s to q found in a DFS of T equals $R[n[e]]$ and that the residue of $n[e]$ is ϵ if $n[e]$ is a final state and otherwise output FALSE. If these conditions are never violated, the algorithm returns TRUE (line 6).

<p align="center">(a)   (b)</p>
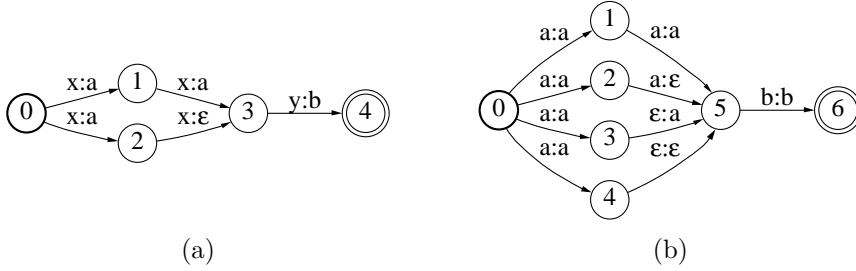
Figure 10: (a) Non-functional finite-state transducer T. (b) Composed transducer $T^{-1} \circ T$. The residue of the path from 0 to 5 through 1 is ϵ and differs from the residue of the path from 0 to 5 through 2, which is a. Thus Identity returns FALSE since two distinct values, ϵ and a, are found for $R[5]$ in a DFS of $T^{-1} \circ T$.

Identity(T) returns TRUE iff the residue of any successful path in T is ϵ. Indeed, Identity(T) returns TRUE iff for any transition e in T, $R[n[e]] = i[e]^{-1}R[p[e]]o[e]$. By induction on the length of π, this is equivalent to: $R[n[\pi]] = i[\pi]^{-1}R[p[\pi]]o[\pi]$ for any path π to a coaccessible state $n[\pi]$. If π is a successful path, $p[\pi] = s$, $R[p[\pi]] = \epsilon$, and the condition can be rewritten as $R[n[\pi]] = i[\pi]^{-1}o[\pi]$. Thus, if the algorithm returns TRUE then $R[n[\pi]] = \epsilon$ for any successful path π and thus $[\![T]\!] = Id_{Dom(T)}$. Conversely, as already pointed out above, if T is the restriction of the identity function to its domain of definition, all paths from s to any coaccessible state q must have the same residue, in particular those found in a DFS of T and $R[n[\pi]] = \epsilon$ for any successful path, thus the algorithm returns TRUE.

Note that if a path π from the initial state to a coaccessible state with a non-pure residue is found, then the residue of a successful path beginning with π cannot be ϵ (see Lemma 3 (iii)). Thus, the algorithm can immediately return FALSE in such cases. Figs. (10)(a)-(b) illustrate the algorithm in the case of a non-functional transducer.

Theorem 7 *There exists an algorithm for testing the identity of a finite-state transducer T in time $O(|E| + |\Delta|\,|Q|)$.*

Proof. Finding the coaccessible states of T and thus defining the array *coacc* can be done in linear time [8]. The algorithm performs a linear number of residue operations: the computation of new residues (line 3) and their comparison (line 4). As mentioned above, we can restrict ourselves to operations on pure residues. In the following, we show that each of these operations can be done in constant time after a pre-processing stage, linear in the size of T.

We first build a deterministic tree $\mathcal{T}$ accepting the input and output strings of all the paths of a DFS tree of the transducer T. A DFS tree of T has $O(|Q|)$ states and edges, thus $\mathcal{T}$ also has $O(|Q|)$ states and edges and we can construct the suffix tree $\mathcal{S}$ of $\mathcal{T}$ in time $O(|\Delta|\,|Q|)$ [12]. Since a DFS is performed in linear time, the total cost of this pre-processing step is $O(|E| + |\Delta|\,|Q|)$.

The residue of a path π is $\langle\pi\rangle = i[\pi]^{-1}o[\pi]$. Thus, when it is pure, it is either a suffix of $o[\pi]$ or the inverse of a suffix of $i[\pi]$. Thus, it can be encoded by a bit

specifying that information and a position in the suffix tree $\mathcal{S}$ corresponding to the string $\langle \pi \rangle$. Two residues are then equal iff they have the same bit and correspond to the same position in the suffix tree. Hence, the comparison of residues can be done in constant time.

The computation of a new residue $a^{-1}rb$ (line 3) from a residue r with a and b in $\Sigma \cup \{\epsilon\}$ can be done in constant time. Indeed, if r is pure, it can be decided in constant time if $a^{-1}rb$ is pure. The corresponding position in $\mathcal{S}$ can be found by going possibly one step up the suffix link followed by one step down in the tree and can therefore be done in constant time. Thus, the total cost of the algorithm Identity is $O(|E| + |\Delta|\,|Q|)$. $\qquad\square$

This result combined with Lemma 5 gives an efficient algorithm for testing the functionality of a transducer.

Corollary 1 *There exists an algorithm for testing the functionality of a finite-state transducer T in time $O(|E|^2 + |\Delta|\,|Q|^2)$.*

Proof. The proof follows immediately Theorem 7, Lemma 5, and the fact that the number of states of $T^{-1} \circ T$ is in $O(|Q|^2)$ and the number of its transitions in $O(|E|^2)$. $\qquad\square$

7. Test of the twins property for finite-state transducers

7.1. Characterization

For finite-state transducers, the twins property can be reformulated in terms of residues of paths in $T^{-1} \circ T$.

Proposition 2 *A transducer T has the twins property iff the following condition holds for the composed transducer $T^{-1} \circ T$: for any path π from an initial state to a cycle c, $\langle \pi \rangle = \langle \pi c \rangle$.*

Proof. Recall from Section (3.5) Eq. (5) defining the twins property for transducers:

$$o[\pi_1]^{-1}o[\pi_1'] = (o[\pi_1]o[\pi_2])^{-1}o[\pi_1']o[\pi_2']$$

By definition of composition, a path from an initial state to a cycle c in $T^{-1} \circ T$ is the result of matching the output of a path π_1 from the initial state to a cycle π_1' and the output of a path π_2 from the initial state to a cycle π_2' in T where c is the result of matching the output of π_1' with the output of π_2'. Thus, the lemma follows immediately the definition of residues of paths and that of the twins property as presented above. $\qquad\square$

Figs. (12)(a)-(b) show example of the composed transducers $T^{-1} \circ T$. Our algorithm for testing the twins property is based on the computation of the residues of paths in $T^{-1} \circ T$ as suggested by the proposition. However, we use two properties of residues to avoid redundant computations. One is the property of Lemma 3 (i), which applies

Figure 11:   (a) Two sibling states q and q' in a transducer T. (b) The corresponding state in $T^{-1} \circ T$. Both the twins property in (a) and the condition on residues in (b) are equivalent to the same condition: $(uv)^{-1}u'v' = u^{-1}u'$.
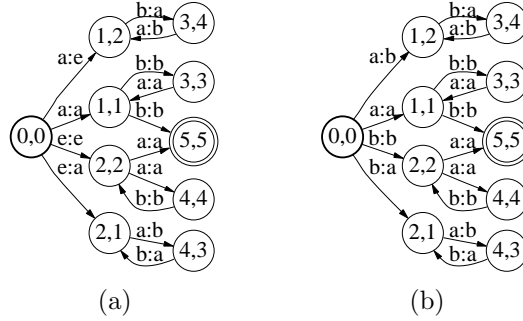

Figure 12: Composed transducer $T^{-1} \circ T$ where T is: (a) the transducer given by Figure 2 (a); (b) the transducer given by Figure 4 (a).

to paths sharing the same suffix, the other is that we only need to compute at most two distinct path residues R_1 and R_2 for any state q, as will be shown by Corollary 2.

In Proposition 2, the condition $\langle \pi \rangle = \langle \pi c \rangle$ implies that $|i[c]| = |o[c]||$ for any cycle c, in particular, $i[c] = \epsilon$ iff $o[c] = \epsilon$. When $i[c] = o[c] = \epsilon$, the condition holds for any path π. A state q is said to be *cycle-accessible* if there exists a path from q to a non-ϵ cycle c ($i[c] \neq \epsilon$ or $o[c] \neq \epsilon$).

Lemma 6 *If T has the twins property, then the residue of any path π_1 in $T^{-1} \circ T$ from the initial state to a cycle-accessible state q is pure.*

Proof. Let π be a path from q to a non-ϵ cycle c. Since T has the twins property, $\langle \pi_1 \pi c \rangle = \langle \pi_1 \pi \rangle$. By Lemma 3 (iv), this implies that $\pi_1 \pi$ is pure, and by the third statement of the same lemma that π_1 is pure.   □

Lemma 7 *Let π_1 and π_2 be two paths leading from the initial state to the same state of a non-ϵ cycle c. Assume that $\langle \pi_1 \rangle = \langle \pi_1 c \rangle$, then $\langle \pi_2 \rangle = \langle \pi_2 c \rangle$ iff $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle \equiv o[c]$.*

Proof. Assume that $\langle \pi_2 c \rangle = \langle \pi_2 \rangle$, then: $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle = o[c]^{-1} \langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle o[c]$. Thus

$\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle$ and $o[c]$ commute. Conversely, if $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle$ and $o[c]$ commute:

$$\langle \pi_2 \rangle = \langle \pi_1 \rangle \langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle = \langle \pi_1 c \rangle \langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle$$
$$= i[c]^{-1} \langle \pi_1 \rangle o[c] \langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle = i[c]^{-1} \langle \pi_1 \rangle \langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle o[c] = \langle \pi_2 c \rangle$$

$\square$

Note that the condition $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle \equiv o[c]$ implies that $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle$ is pure since $o[c]$ is in Σ^*.

Corollary 2 *Let π_1, π_2, and π be three paths leading from the initial state to the same state of a non-ϵ cycle c such that $\langle \pi_1 \rangle \neq \langle \pi_2 \rangle$. Assume that $\langle \pi_1 \rangle = \langle \pi_1 c \rangle$ and $\langle \pi_2 \rangle = \langle \pi_2 c \rangle$, then: $\langle \pi \rangle = \langle \pi c \rangle$ iff $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle \equiv \langle \pi_1 \rangle^{-1} \langle \pi \rangle$.*

Proof. By Lemma 7, $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle \equiv o[c]$ and $\langle \pi \rangle = \langle \pi c \rangle$ iff $\langle \pi_1 \rangle^{-1} \langle \pi \rangle \equiv o[c]$. Thus, if $\langle \pi \rangle \neq \langle \pi_1 \rangle$, by transitivity of $\equiv$ over $\Sigma^{(*)} - \{\epsilon\}$, $\langle \pi \rangle = \langle \pi c \rangle$ iff $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle \equiv \langle \pi_1 \rangle^{-1} \langle \pi \rangle$. If $\langle \pi \rangle = \langle \pi_1 \rangle$, then by hypothesis, $\langle \pi_1 \rangle = \langle \pi_1 c \rangle$, that is $\langle \pi \rangle = \langle \pi c \rangle$. And $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle \equiv \langle \pi_1 \rangle^{-1} \langle \pi \rangle$ trivially holds since $\langle \pi_1 \rangle^{-1} \langle \pi \rangle = \epsilon$. $\square$

Lemma 8 *If T has the twins property, then the following condition holds for the composed transducer $T^{-1} \circ T$: let π_1, π_2 and π_3 be three paths leading to the same cycle-accessible state, then $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle \equiv \langle \pi_1 \rangle^{-1} \langle \pi_3 \rangle$.*

Proof. The lemma is a direct consequence of Corollary 2 and Lemma 3(ii). $\square$

By proposition 2, testing the twins property is equivalent to testing that $\langle \pi \rangle = \langle \pi c \rangle$ for any path π from an initial state to a cycle c in the transducer $U = T^{-1} \circ T$. We can limit ourselves to the cycle-accessible part of U since the condition is otherwise always satisfied.

The main idea behind the design of our algorithm for testing the twins property for transducers is based on Corollary 2. It consists of computing, when possible, two distinct residues for each cycle-accessible state q of the transducer $U = T^{-1} \circ T$. These residues correspond to two paths π_1 and π_2 from the initial state to q. We will be testing $\langle \pi_1 \rangle^{-1} \langle \pi_2 \rangle \equiv \langle \pi_1 \rangle^{-1} \langle \pi \rangle$ rather than $\langle \pi \rangle = \langle \pi c \rangle$ since the former can be tested more efficiently and we will use lemma 8 to restrict the number of paths π to consider.

We need to compute the distinct residues in a consistent and efficient manner: if the path corresponding to the first (second) residue computed for a state q is $\pi = \pi' e$, then the path corresponding to the first (resp. second) residue computed for $p[e]$ must be π'. Thus, the set of paths corresponding to the first (or second) residue computed for each state forms a spanning tree of $U = T^{-1} \circ T$.

Let T_1 and T_2 be two spanning trees of U. For each state q, we denote by $\Pi_k[q]$ the path in T_k, $(k = 1, 2)$, leading to q from the initial state and we denote by σ_k the predecessor function of the tree T_k: $\sigma_k[q]$ is the predecessor node of q in T_k. We say that T_1 and T_2 are *consistent* if for any cycle-accessible state q, the paths $\Pi_1[q]$ and $\Pi_2[q]$ are pure and if $\langle \Pi_1[q] \rangle \neq \langle \Pi_2[q] \rangle$ when q can be reached by two paths with distinct residues. In the following, we denote by $R_k[q]$ the residue $\langle \Pi_k[q] \rangle$ and denote by $scc[q]$ the strongly connected component of a state q of $T^{-1} \circ T$.

Lemma 9 *If T has the twins property, then the composed transducer $U = T^{-1} \circ T$ admits two consistent spanning trees T_1 and T_2.*

Proof. Our proof is constructive. We denote by s the initial state of U and set $\sigma_k[s] = \text{NIL}$. We assume that the cycle-accessible states q of U have been marked by $cyacc[q] = \text{TRUE}$ and that the attributes R_1 and R_2 have been initialized to an undefined value ∞. This can be done in linear time in the size of U.

The following is the pseudocode of an algorithm for computing two consistent spanning trees T_1 and T_2 for the transducer U if we complete σ_2 by setting $\sigma_2[q] = \sigma_1[q]$ for all states q for which $\sigma_2[q]$ has not been defined at the end of the execution of the algorithm.

SpanningTrees(q, k)
    1   **for** each $e \in E[q]$ such that $cyacc[n[e]] = \text{TRUE}$
    2   **do** $R \leftarrow i[e]^{-1} R_k[q] \, o[e]$
    3   **if** $(k = 1$ **and** $R_1[n[e]] \neq \infty$ **and** $R_2[n[e]] = \infty$ **and** $R_1[n[e]] \neq R)$
    4   **then** $R_2[n[e]] \leftarrow R; \, \sigma_2[n[e]] \leftarrow q$
    5   SpanningTrees$(n[e], 2)$
    6   **if** $(R_k[n[e]] = \infty)$
    7   **then** $R_k[n[e]] \leftarrow R; \, \sigma_k[n[e]] \leftarrow q$
    8   SpanningTrees$(n[e], k)$

If we leave aside the instructions of lines 3-5, SpanningTrees(q, k) corresponds to the basic DFS algorithm applied to the part of U that is accessible and cycle-accessible and that we will refer to as U'. Thus, it creates a DFS tree T_k and computes R_k in a consistent manner, that is: $R_k[n[e]] = i[e]^{-1} R_k[p[e]] o[e]$ (lines 2 and 7).

The initial call is SpanningTrees$(s, 1)$, which creates a spanning tree T_1 that coincides with the DFS tree of U' rooted in s. When a new residue is found for a state q for which $R_1[q]$ is already defined and $R_2[q]$ is not, that residue is assigned to $R_2[q]$ and T_2 is constructed by starting a DFS of U' from q (lines 3-5). We will refer to such states q as the *root states*. The computation of the residue R_2 is consistent, that is $R_2[n[e]] = i[e]^{-1} R_2[p[e]] o[e]$ for a transition e in T_2.

Thus, a residue $R_2[q]$ is computed either at a root state, or when q belongs to a DFS tree rooted in a root state r. In the former case, by construction, $R_2[q] \neq R_1[q]$. In the latter, there is a tree path π in both T_1 and T_2 from r to q. Since the computation of the residues along the tree paths is consistent $R_1[q] = i[\pi]^{-1} R_1[r] o[\pi]$ and $R_2[q] = i[\pi]^{-1} R_2[r] o[\pi]$, which implies $R_2[q] \neq R_1[q]$ since $R_2[r] \neq R_1[r]$. Thus, in all cases, when $R_2[q]$ is defined, $R_2[q] \neq R_1[q]$.

By Lemma 6, since T has the twins property, all the residues computed by the algorithm are pure. To show that T_1 and T_2 are consistent, we need to show that when $R_2[q]$ is undefined for a state q, then there exists no path π from s to q with a residue distinct from $R_1[q]$. Assume that this does not hold for a state q and let π be a path from s to q with $\langle \pi \rangle \neq R_1[q]$. Without loss of generality, we can assume that q is the first state along π with that property. Denote by e the last transition of π. Let π' be defined by $\pi = \pi' e$ and q' by $q' = n[\pi']$.

$R_2[q']$ must be undefined since otherwise $R_2[q]$ would have been computed by the call SpanningTrees$(q', 2)$ of line 5 or 7. During the execution of SpanningTrees$(q', 1)$, when the transition e is selected, $R_1[q]$ is already defined since otherwise it would be set to $R = i[e]^{-1}R_1[q']o[e] = \langle \pi \rangle \neq R_1[q]$. Thus, the condition of line 3 holds and $R_2[q]$ is set to $\langle \pi \rangle$ (line 4), which leads to a contradiction. $\qquad\square$

Our algorithm for testing the twins property is derived from the following theorem which gives a necessary and sufficient condition based on two consistent spanning trees T_1 and T_2 of U for T to have the twins property.

Theorem 8 *A transducer T has the twins property iff the following condition holds for the composed transducer $U = T^{-1} \circ T$: there exist two consistent spanning trees T_1 and T_2 such that for any edge e such that $n[e]$ is cycle-accessible:*

(i) *if $scc[n[e]] = scc[p[e]]$ then for $k = 1, 2$, $R_k[n[e]] = i[e]^{-1}R_k[p[e]]o[e]$.*

(ii) *if $scc[n[e]] \neq scc[p[e]]$ then for any residue R in $\{\langle \Pi_1[p[e]]\,e\rangle, \langle \Pi_2[p[e]]\,e\rangle\}$, $R_1[n[e]]^{-1}R_2[n[e]] \equiv R_1[n[e]]^{-1}R$.*

Proof. Assume that T has the twins property. Then, by Lemma 9, U admits two consistent spanning trees T_1 and T_2. Let e be an edge within a strongly connected component S ($scc[n[e]] = scc[p[e]]$). Let k be fixed and let q be the first state examined in the computation of the spanning tree T_k that belongs to S. Since in the construction of the spanning trees of Lemma 9 the states of U are visited according to a depth-first search order, T_k contains a path π from q to $p[e]$ and a path π' from q to $n[e]$. Since q and $n[e]$ are both in S, there exists a path π'' in U from $n[e]$ to q. Thus $\pi\,e\,\pi''$ and $\pi'\pi''$ are both cycles at q and in view of proposition 2: $\langle \Pi_k[q]\rangle = \langle \Pi_k[q]\pi'\pi''\rangle = \langle \Pi_k[q]\pi\,e\,\pi''\rangle$, which implies:

$$\langle \Pi_k[q]\pi'\rangle = \langle \Pi_k[q]\pi\,e\rangle \tag{6}$$

Since T_k is consistent, $R_k[n[e]] = \langle \Pi_k[q]\pi'\rangle$ and $R_k[p[e]] = \langle \Pi_k[q]\pi\rangle$ and Eq. (6) can be rewritten as: $R_k[n[e]] = i[e]^{-1}R_k[p[e]]o[e]$. This proves the first statement of the theorem. The second statement of the theorem follows directly Lemma 8.

Conversely, assume that U verifies the conditions of the theorem. Note that (i) trivially implies that (ii) holds for any edge e including when $scc[n[e]] = scc[p[e]]$. By induction on the length of π, (i) also implies that for any path π such that $scc[p[\pi]] = scc[n[\pi]]$ and for $k = 1, 2$:

$$R_k[n[\pi]] = \langle \Pi_k[p[\pi]]\pi\rangle \tag{7}$$

We will first show that for any cycle-accessible state q and any path π from the initial state to q, we have:

$$R_1[q]^{-1}R_2[q] \equiv R_1[q]^{-1}\langle \pi\rangle \tag{8}$$

Assume that the strongly connected components of U are numbered in topological order, thus for any path π, $scc[p[\pi]] \leq scc[n[\pi]]$. This ordering can be computed in linear time using classical graph algorithms [8]. We will show that Eq. (8) holds for any q by induction on $scc[q]$.

For the basis, note that the initial state s belongs to the first strongly connected component, thus, if π is a path from s to a state q with $scc[q] = 0$, then $scc[p[\pi]] = scc[n[\pi]] = 0$. By Eq. (7), $R_1[n[\pi]] = \langle \Pi_1[p[\pi]]\pi \rangle$, that is $R_1[q] = \langle \pi \rangle$, in which case Eq. (8) trivially holds.

Assume now that Eq. (8) holds for any state q with $scc[q] < n$, $n \geq 1$. Let q be a cycle-accessible state with $scc[q] = n$ and let π be a path from the initial state to q. π can be factorized as $\pi = \pi' e \pi''$ with $scc[p[e]] < scc[n[e]] = n$. Let us first show that:

$$R_1[n[e]]^{-1} R_2[n[e]] \equiv R_1[n[e]]^{-1} \langle \pi' e \rangle \tag{9}$$

Since T_1 and T_2 are consistent, if $R_1[p[e]] = R_2[p[e]]$, then all paths from the initial state to $p[e]$ have the same residue, in particular: $\langle \pi' \rangle = R_1[p[e]] = R_2[p[e]]$, which implies: $\langle \pi' e \rangle = \langle \Pi_1[p[e]]e \rangle = \langle \Pi_2[p[e]]e \rangle$. Eq. (9) then coincides with the statements (ii) of the theorem. If $R_1[p[e]] \neq R_2[p[e]]$, then Eq. (9) results from the application of Lemma 4 with $r_k = R_k[n[e]]$ and $s_k = \langle \Pi_k[p[e]]e \rangle$ for $k = 1, 2$, and $r = \langle \pi' e \rangle$ to the equations of statement (ii):

$$R_1[n[e]]^{-1} R_2[n[e]] \equiv R_1[n[e]]^{-1} \langle \Pi_k[p[e]]\, e \rangle$$

and the induction hypothesis:

$$R_1[p[e]]^{-1} R_2[p[e]] \equiv R_1[p[e]]^{-1} \langle \pi' \rangle$$

with $s_1^{-1} s_2 = R_1[p[e]]^{-1} R_2[p[e]] \neq \epsilon$. By Lemma 3 (ii), Eq. (9) implies:

$$\langle \Pi_1[n[e]]\pi'' \rangle^{-1} \langle \Pi_2[n[e]]\pi'' \rangle \equiv \langle \Pi_1[n[e]]\pi'' \rangle^{-1} \langle \pi \rangle$$

Since $scc[p[\pi'']] = scc[n[\pi'']] = n$, by Eq. (7), $\langle \Pi_k[n[e]]\pi'' \rangle^{-1} = R_k[n[\pi'']] = R_k[q]$, for $k = 1, 2$. This proves Eq. (8).

We can now prove that T has the twins property. Let q be a state with a non-ϵ cycle c, and let π be a path leading to q from the initial state. As already mentioned, (i) implies that $\langle \Pi_k[q]\, c \rangle = \langle \Pi_k[q] \rangle$ holds for $k = 1, 2$. Since the spanning trees T_k are consistent, if $R_1[q] = R_2[q]$ then all paths leading to q have the same residue, in particular: $\langle \pi c \rangle = \langle \pi \rangle$. Otherwise if $R_1[q] \neq R_2[q]$, by Corollary 2, Eq. (8) implies that $\langle \pi c \rangle = \langle \pi \rangle$, which, in view of proposition 2, shows that T has the twins property. $\qquad \square$

7.2. Algorithm

The theorem leads to a short and simple algorithm for testing the twins property. The pseudocode given in the proof of Lemma 9 is augmented to verify the conditions of the theorem. Since we are not interested in the actual definition of the spanning trees T_1 and T_2 but only in the residues at each node of those trees, we can omit the definition of the predecessor functions of T_1 and T_2 in the algorithm. The following is the pseudocode of our algorithm.

```
Residue(q, k)
 1   for each e ∈ E[q]  such that cyacc[n[e]] = TRUE
 2       do  R ← i[e]⁻¹ Rₖ[q] o[e]
 3           if (R ∉ Σ* ∪ (Σ⁻¹)*) then return FALSE
 4           if (scc[n[e]] = scc[q])
 5               then  if (Rₖ[n[e]] ≠ ∞  and  Rₖ[n[e]] ≠ R) return FALSE
 6           else  if (k = 1 and R₁[n[e]] ≠ ∞ and R₂[n[e]] = ∞ and R₁[n[e]] ≠ R)
 7                   then   R₂[n[e]] ← R
 8                          Residue(n[e], 2)
 9           else  if (R₁[n[e]] ≠ ∞ and R₂[n[e]] ≠ ∞)
10                   then   if (R₁[n[e]]⁻¹R₂[n[e]] ≢ R₁[n[e]]⁻¹R)
11                            then   return FALSE
12           if (Rₖ[n[e]] = ∞)
13               then   Rₖ[n[e]] ← R
14                      Residue(n[e], k)
15  return TRUE
```

We use a DFS of U to compute the residues R_1 and R_2 which are initialized to an undefined value ∞ starting from the initial state and with $k = 1$. Thus the original call is Residue$(s, 1)$ where s is the initial state. We also assume that the cycle-accessible states q of U have been marked by $cyacc[q] = $ TRUE, which can be done in time linear in the size of U.

According to the theorem, the search can be limited to the edges e such that $n[e]$ is cycle-accessible (line 1). The new residue $R = i[e]^{-1}R_k[p[e]]o[e]$ is computed in line 2 and its purity checked in line 3. Lines 4-5 correspond exactly to the condition (i) of the theorem. When $R_1[n[e]]$ has already been computed and $R \neq R_1[n[e]]$, $R_2[n[e]]$ is set to R and the construction of the second spanning tree T_2 continues (lines 7-8). Lines 9-11 correspond exactly to the second condition of the theorem. When $R_k[n[e]]$ is undefined, it is set to R and the construction of T_k and R_k continues with the call Residue$(n[e], k)$. The algorithm returns TRUE when the conditions of the theorem have been verified (line 15).

Each edge e is traversed at most two times during the execution of the algorithm since Residue(q, k) is called only when $R_1[q]$ or $R_2[q]$ equals ∞. This guarantees the termination of the algorithm. Its correctness follows immediately theorem 8.

Several operations on strings are performed in each execution of the loop of lines 1-14: computation of a new residue, comparison of two residues, and comparison of the primitive root of two residues. Some of these operations can be performed very efficiently, e.g., by using a suffix tree for the representation of the residues. However, the comparison of the primitive roots dominates the complexity of these operations. In the worst case, the comparison of the primitive roots may require up to $|Q|^2 - 1$ comparisons, the maximal length of a string of a spanning tree. Since each transition of U is visited at most two times and since the size of U is $O(|Q|^2 + |E|^2))$, the total complexity of the algorithm is $O(|Q|^2 (|Q|^2 + |E|^2))$, based on our current analysis of the complexity of the comparison of primitive roots. Any improvement to the worst cost complexity of this operation would lead to an improvement of the complexity of our algorithm.
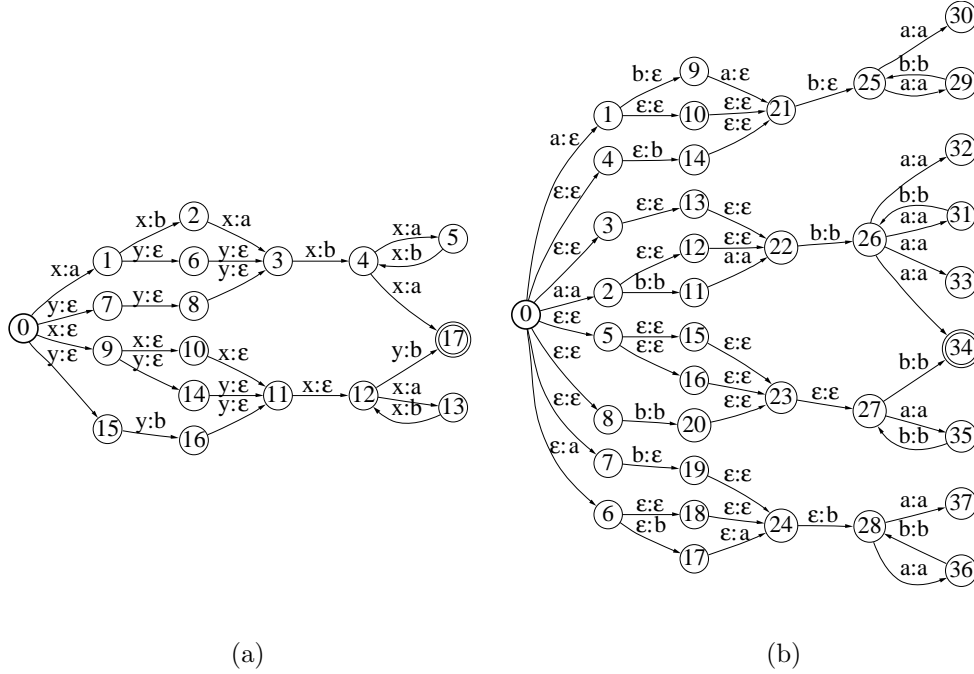
(a)   (b)

Figure 13: (a) A finite-state transducer T. (b) The composed machine $U = T^{-1} \circ T$.

7.3. Example

In this section, we are illustrating the execution of the algorithm in a particular example.

Let T be the finite-state transducer shown in Fig. (13) (a). Fig. (13) (b) shows the corresponding composed transducer $U = T^{-1} \circ T$. We can distinguish three distinct sets in U: (I) the states and transitions accessible from states 1 and 4; (II) the states and transitions accessible from 2, 3, 5 and 8; and (III) the states and transitions accessible from 6 and 7. In the set (II), all transitions have the same input and output label and the residue of any path π from 0 to a state in (II) is ϵ. Thus, the condition $\langle \pi c \rangle = \langle \pi \rangle$ clearly holds for such a path. Set (III) is simply the symmetric of set (I): it is obtained form (I) by exchanging the input and output labels of each transition. Thus, to examine the execution of our algorithm, we can restrict ourselves to part (I).

The residues R_1 and R_2 computed by the algorithm are shown by Fig. (14). Note that state 30 is never visited since it is not cycle-accessible. Fig. (15) shows the tree of function calls restricted to the set (I). A full arrow corresponds to the execution of the instruction of line 14, a dotted arrow to that of the instruction of line 8.

A dashed box represents a call of Residue(q, k) in which the condition of line 5 is tested. A dotted box represents a call in which the condition of line 10 is tested. For example, in the execution of the call Residue$(14, 1)$, the condition of line 10 is tested

q	0	1	4	9	10	14	21	25	29
R_1	ϵ	a^{-1}	ϵ	$b^{-1}a^{-1}$	a^{-1}	b	$a^{-1}b^{-1}a^{-1}$	$b^{-1}a^{-1}b^{-1}a^{-1}$	$a^{-1}b^{-1}a^{-1}b^{-1}$
R_2	∞	∞	∞	∞	∞	∞	a^{-1}	$b^{-1}a^{-1}$	$a^{-1}b^{-1}$

Figure 14: Residues R_1 and R_2 computed by the algorithm for the states in set (I) of the transducer U shown in Fig. (13) (b).
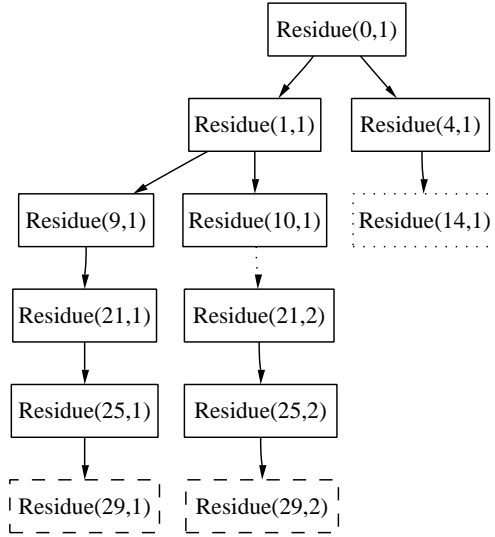

Figure 15: The portion of the tree of function calls Residue(q, k) restricted to the states in set (I) starting from the initial state $q = 0$.

for the transition $e = (14, \epsilon, \epsilon, 21)$, with $R = \epsilon^{-1} R_1[14] \, \epsilon = b$. We have:

$$R_1[21]^{-1} R_2[21] = (a^{-1}b^{-1}a^{-1})^{-1} a^{-1} = abaa^{-1} = ab$$

$$R_1[21]^{-1} R = (a^{-1}b^{-1}a^{-1})^{-1} b = abab$$

Since $ab \equiv abab$, the condition of line 10 holds. More generally, the execution of the algorithm shows that the set (I) satisfies the conditions of Theorem 8. Since sets (III) and (I) are symmetric, this also holds for set (III). Set (II) also trivially satisfies these conditions, thus T has the twins property.

8. Experimental Results

We tested the efficiency of our algorithms by running them with very large automata and transducers found in large-vocabulary speech recognition systems on a Pentium III 700 MHz with a cache size of 2048 Kb. The results showed that they can be used to test the determinizability of very large machines in short time. As an example, it

took about 1m30s to test the twins property for a weighted automaton A of about 3.9M states and 7.6M transitions, with about $118,000$ strongly connected components. The intersection machine $A \cap A^{-1}$ had about 6M states, 11M transitions and $144,000$ strongly connected components.

9. Conclusion

We presented new and efficient algorithms for testing the twins property for weighted automata and finite-state transducers and for testing the functionality of finite-state transducers. Our implementation of the algorithms shows these algorithms to be practical even for very large automata and transducers. They can be used to test the determinizability of finite-state transducers and that of cycle-unambiguous weighted automata and weighted transducers.

Acknowledgements

References

[1] A. V. AHO, R. SETHI, AND J. D. ULLMAN, *Compilers, Principles, Techniques and Tools*, Addison Wesley: Reading, MA, 1986.

[2] C. ALLAUZEN AND M. MOHRI, *p-Subsequential Transducers*, in Seventh International Conference, CIAA 2002, Jean-Marc Champarnaud and Denis Maurel, ed., vol. to appear of Lecture Notes in Computer Science, Tours, France, July 2002, Springer-Verlag, Berlin-NY.

[3] M.-P. BÉAL, O. CARTON, C. PRIEUR, AND J. SAKAROVITCH, *Squaring transducers: An efficient procedure for deciding functionality and sequentiality*, in Proceedings of LATIN'2000, vol. 1776 of Lecture Notes in Computer Science, Springer, 2000.

[4] J. BERSTEL, *Transductions and Context-Free Languages*, Teubner Studienbucher: Stuttgart, 1979.

[5] A. L. BUCHSBAUM, R. GIANCARLO, AND J. R. WESTBROOK, *On the Determinization of Weighted Finite Automata*, SIAM Journal of Computing, 30 (2000), pp. 1502–1531.

[6] C. CHOFFRUT, *Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles*, Theoretical Computer Science, 5 (1977), pp. 325–338.

[7] ——, *Contributions à l'étude de quelques familles remarquables de fonctions rationnelles*, PhD thesis, (thèse de doctorat d'Etat), Université Paris 7, LITP: Paris, France, 1978.

[8] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, The MIT Press: Cambridge, MA, 1992.

[9] K. CULIK II AND J. KARI, *Digital Images and Formal Languages*, in Handbook of Formal Languages, G. Rozenberg and A. Salomaa, eds., vol. 3, Springer, 1997, pp. 599–616.

[10] F. GIRE, *Two decidable problems for infinite words*, Information Processing Letter, 22 (1986), pp. 135–140.

[11] E. M. GURARI AND O. H. IBARRA, *A note on finite-valued and finitely ambiguous transducers*, Mathematical System Theory, 16 (1983), pp. 61–66.

[12] S. INENAGA, H. HOSHINO, A. SHINOHARA, M. TAKEDA, AND S. ARIKAWA, *Construction of the CDAWG for a Trie*, in Proceedings of the Prague Stringology Conference (PSC'01), Czech Technical University, 2001.

[13] W. KUICH AND A. SALOMAA, *Semirings, Automata, Languages*, no. 5 in EATCS Monographs on Theoretical Computer Science, Springer-Verlag, Berlin, Germany, 1986.

[14] M. MOHRI, *Finite-State Transducers in Language and Speech Processing*, Computational Linguistics, 23 (1997).

[15] ——, *Minimization Algorithms for Sequential Transducers*, Theoretical Computer Science, 234 (2000), pp. 177–201.

[16] M. MOHRI, F. C. N. PEREIRA, AND M. RILEY, *Weighted Automata in Text and Speech Processing*, in Proceedings of the 12th biennial European Conference on Artificial Intelligence (ECAI-96), Workshop on Extended finite state models of language, Budapest, Hungary, ECAI, 1996.

[17] D. PERRIN, *Words*, in Combinatorics on words, M. Lothaire, ed., Cambridge Mathematical Library, Cambridge University Press, 1997.

[18] M. P. SCHÜTZENBERGER, *Sur les relations rationnelles entre monoides libres*, Theoretical Computer Science, 3 (1977), pp. 243–259.

[19] ——, *Sur une variante des fonctions séquentielles*, Theoretical Computer Science, 4 (1977), pp. 47–57.

[20] A. WEBER AND R. KLEMM, *Economy of Description for Single-Valued Transducers*, Information and Computation, 118 (1995), pp. 327–340.