International Journal of Foundations of Computer Science
© World Scientific Publishing Company

On the Disambiguation of Finite Automata and Functional Transducers

MEHRYAR MOHRI
Courant Institute of Mathematical Sciences,
251 Mercer Street, New York, NY 10012, US,
and
Google Research,
76 Ninth Avenue, New York, NY 10011, US,
mohri@cs.nyu.edu

This paper introduces a new disambiguation algorithm for finite automata and functional finite-state transducers. It gives a full description of this algorithm, including a detailed pseudocode and analysis, and several illustrating examples. The algorithm is often more efficient and the result dramatically smaller than the one obtained using determinization for finite automata or the construction of Schützenberger. The unambiguous automaton or transducer created by our algorithm are never larger than those generated by the construction of Schützenberger. In fact, in a variety of cases, the size of the unambiguous transducer returned by our algorithm is only linear in that of the input transducer while the transducer created by the construction of Schützenberger is exponentially larger. Our algorithm can be used effectively in many applications to make automata and transducers more efficient to use.

Keywords: finite automata; finite-state transducers; disambiguation; determinization.

1. Introduction

Finite automata and transducers are used in a variety of applications in text and speech processing [10, 13], bioinformatics [8], image processing [1], optical character recognition [6], and many others. In these applications, the automata and transducers obtained as a result of various complex operations may not be efficient to use. Some optimization algorithms such as determinization can make their use more time-efficient. However, the result of determinization is sometimes prohibitively large and not all finite-state transducers are determinizable [7, 11].

This paper presents and analyzes an alternative optimization algorithm, *disambiguation*, which in practice can have efficiency benefits similar to determinization. Our disambiguation algorithm is novel and applies to finite automata, including automata with ϵ-transitions, and to *functional finite-state transducers*, that is those representing a partial function. Disambiguation returns an automaton or transducer equivalent to the input that is *unambiguous*, that is one that admits no two accepting paths labeled with the same (input) string. In many instances, the absence of ambiguity can be useful to make search more efficient by reducing the number of paths to explore for very large automata or transducers with several hundred thousand or millions of transitions in text and speech processing or in bioinformatics, and there are many other critical needs for the disambiguation of automata and transducers.

For finite automata, one way to proceed to obtain an unambiguous and equivalent automaton is simply to apply the standard determinization algorithm. For some non-deterministic finite automata of size n the size of an equivalent unambiguous automaton it at least $\Omega(2^{\sqrt{n}})$ [16], which gives a lower bound on the time and space complexity of any disambiguation algorithm for finite automata. But, as we shall see, for some input automata our algorithm can take exponentially less time than determinization and return an equivalent unambiguous automaton exponentially smaller than the one obtained by using determinization.

For finite-state transducers, disambiguation applies to a broader set of transducers than those that can be determinized using the algorithm described in [11] since it applies to any functional transducer. In contrast, it was shown by [3] that a functional transducer is determinizable if and only if it additionally verifies the *twins property* [7, 11, 2].

Our disambiguation algorithm is also often dramatically more efficient and results in substantially smaller transducers than those obtained using a disambiguation algorithm based on a construction of Schützenberger [17, 15], also described by E. Roche and Y. Schabes in the introductory chapter of [14]. In particular, when the input transducer is unambiguous, our algorithm simply returns the same transducer, while the result of the algorithm presented in [14] can be exponentially larger.

The remainder of this paper is organized as follows. In Section 2, we introduce the notation and basic concepts needed for the presentation and analysis of our algorithm. In Section 3, we present our disambiguation algorithm for finite automata in detail, including the proof of its correctness. Section 4 gives a brief description of the extension of our algorithm to the case of finite automata with ϵ-transitions. In Section 5, we show how the algorithm can be be used to disambiguate functional transducers and illustrate it with several examples.

2. Preliminaries

We will denote by ϵ the empty string. A finite automaton A with ϵ-transitions is a 5-tuple (Σ, Q, I, F, E) where Σ is a finite alphabet, Q a finite set of states, $I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states, and E a finite multiset of transitions, which are elements of $Q \times (\Sigma \cup \{\epsilon\}) \times Q$. We denote by $|A| = |Q| + |E|$ the *size of an automaton* A, that is the sum of the number states and transitions defining A.

A path π of an automaton is an element of E^* with consecutive transitions. The label of a path is the string obtained by concatenation of the labels of its constituent transitions. We denote by $P(p, x, q)$ the set of paths from p to q labeled with x or, more generally, by $P(R, x, R')$ the set of paths labeled with x from some set of states R to some set of states R'. We also denote by $P(R, R')$ the set of all paths from R to R'. An *accepting path* is an element of $P(I, F)$. The *language accepted by an automaton* A is the set of strings labeling its accepting paths and is denoted by $L(A)$. Two automata A and B are said to be equivalent when $L(A) = L(B)$.

We will say that a state p *can be reached by a string* x when there exists a path from an initial state to p labeled with x. When two states can be reached by the same string, we say that they are *co-reachable*. We will also say that two states p and q *share a common future* when they admit a common string x to reach a final state, that is when there exists a string x such that $P(p, x, F) \neq \emptyset$ and $P(q, x, F) \neq \emptyset$. For any subset $s \subseteq Q$ and $x \in \Sigma^*$, we will denote by $\delta^*(s, x)$ the set of states that can be reached from the states in s by a path labeled with x.

A *finite-state transducer* is a finite automaton in which each transition is augmented with an output label, which is an element of $(\Delta \cup \{\epsilon\})$, where Δ is a finite alphabet. For any transducer T, we denote by T^{-1} its *inverse*, that is the transducer obtained from T by swapping the input and output label of each transition.

We will use the standard algorithm to compute the intersection $A \cap A'$ of two automata A and A' [12], whose states are pairs formed by a state of A and a state of A', and whose transitions are of the form $((p, q), a, (p', q'))$, where (p, a, q) is a transition in A and (p', a, q') in A'.

An automaton A is said to be *trim* if all of its states lie on some accepting path. It is said to be *unambiguous* if no string $x \in \Sigma^*$ labels two distinct accepting paths, *finitely ambiguous* if there exists $k \in \mathbb{N}$ such that no string labels more than k accepting paths, *polynomially ambiguous* if there exists a polynomial P with coefficients in $\mathbb{N}$ such that no string x labels more than $P(|x|)$ accepting paths. The finite, polynomial, and exponential ambiguity of an automaton with ϵ-transitions can be tested in polynomial time [4].

3. Disambiguation algorithm for finite automata

In this section, we describe in detail our disambiguation algorithm for finite automata. The algorithm is first described for automata without ϵ-transitions. The extension to the case of automata with ϵ-transitions is discussed later. Our algorithm in general does not require a full determinization. In fact, in some cases where determinization creates 2^n states where n is the number of states of the input automaton, the size of the automaton created by our algorithm is only $O(n)$.

3.1. *Description*

Figure 1 gives the pseudocode of the algorithm. The first step of the algorithm consists of computing the automaton $A \cap A$ and of trimming it by removing non-coaccessible states (line 1). The cost of this computation is in $O(|A|^2)$ since the complexity of intersection is quadratic and since trimming can be done in linear time. The automaton B thereby constructed can be used to determine in constant time if two states q and r of A that can be reached from I via the same string share

```
Disambiguation($A$)
 1  $B \leftarrow$ Trim($A \cap A$)
 2  for each $i \in I$ do
 3      $s \leftarrow \{i' \colon i' \in I \wedge (i, i') \in B\}$
 4      $I' \leftarrow Q' \leftarrow Q' \cup \{(i, s)\}$
 5      Enqueue($\mathcal{Q}, (i, s)$)
 6  for each $(u, u') \in I'^2$ do
 7      $R \leftarrow R \cup \{(u, u')\}$
 8  while $\mathcal{Q} \neq \emptyset$ do
 9      $(p, s) \leftarrow$ Head($\mathcal{Q}$)
10      Dequeue($\mathcal{Q}$)
11      if $\big((p \in F)$ and $(\nexists (p', s') \in F'$ with $(p', s')\ R\ (p, s))\big)$ then
12          $F' \leftarrow F' \cup \{(p, s)\}$
13      for each $(p, a, q) \in E$ do
14          $t \leftarrow \{r \in \delta^*(s, a) \colon (q, r) \in B\}$
15          if $\big(\nexists ((p', s'), a, (q, t)) \in E'$ with $(p', s')\ R\ (p, s)\big)$ then
16              if $((q, t) \notin Q')$ then
17                  $Q' \leftarrow Q' \cup \{(q, t)\}$
18                  Enqueue($\mathcal{Q}, (q, t)$)
19              $E' \leftarrow E' \cup \big\{\big((p, s), a, (q, t)\big)\big\}$
20              for each $(p', s')$ such that $\big((p', s') R\ (p, s)\big)$ and $\big((p', s'), a, (q', t')\big) \in E'$ do
21                  $R \leftarrow R \cup \{(q, t), (q', t'))\}$
22  return $A'$
```

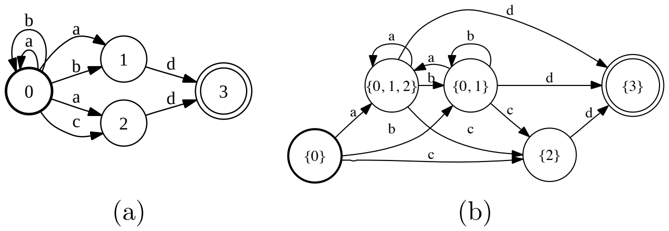
Fig. 1. New disambiguation algorithm for finite automata. Q', I', F', and E' are initialized to $\emptyset$.

a common future simply by checking if (q, r) is a state of B. Indeed, by definition of intersection, this property holds iff (q, r) is a state of B. As shown by the following proposition, the automaton B is in fact directly related to the ambiguity of A.

Proposition 1 ([4]) *Let A be a trim finite automaton with no ϵ-transition. A is unambiguous iff no coaccessible state in $A \cap A$ is of the form (p, q) with $p \neq q$.*

Proof. Since A is trim, the states of $A \cap A$ are all accessible by construction. Thus, a state (p, q) in $A \cap A$ is coaccessible iff it lies on an accepting path, that is by definition of intersection, iff there are two paths $\pi = \pi_1 \pi_2 \in P(I, F)$ and $\pi' = \pi'_1 \pi'_2 \in P(I, F)$ with $\pi_1 \in P(I, p)$ and $\pi'_1 \in P(I, q)$, with π_1 and π'_1 sharing the same label and π_2 and π'_2 also sharing the same label. Thus, A is unambiguous iff $p = q$. □

The algorithm constructs an unambiguous automaton $A' = (Q', E', I', F')$. The set of states Q' are of the form (p, s) where p is a state of A and s a subset of the states of A. Line 2 defines the initial states which are of the form (i, s) with $i \in I$ and s a



(a) (b)

Fig. 2. (a) Automaton A. (b) Result of determinization applied to A.

subset of the states in I sharing a common future with i. The algorithm maintains a relation R such that two states of A' are in relation via R iff they can be reached by the same string from the initial states. In particular, since all initial states are reachable by ϵ, any two pair of initial states are in relation via R (lines 6-7).

The algorithm also maintains a queue $\mathcal{Q}$ containing the set of states (p, s) of Q' left to examine and for which the outgoing transitions are to be determined. We will assume that the co-reachability relation R has been precomputed, using for example the standard composition algorithm, in a first pass in which all states (p, s) have been visited in some order. Then, the queue discipline, that is the order in which states are added or extracted from $\mathcal{Q}$ is arbitrary and does not affect the correctness of the algorithm. However, different orderings can result in different but equivalent resulting automata.

At each execution of the loop of lines 8-21, a new state (p, s) is extracted from $\mathcal{Q}$ (lines 9-10). To avoid an ambiguity due to finality, state (p, s) is made final only if there is no final state $(p', s') \in F'$ in relation with (p, s) (lines 11-12).

Each outgoing transition (p, a, q) of p is then examined. Line 14 defines t to be the subset of the states of A that can be reached from a state of s by reading x but excludes states q' that do not share a common future with q. This is because the subsets are used to detect ambiguities. If q and q' do not share a common future even though there are paths with the same label x reaching them, these paths cannot be completed to reach a final state with the same label. Thus, if X is the set of strings leading to a state (p, s) of Q', the subset s contains exactly the set of states r of A that can be reached via X from I and that share a common future with p.

To avoid creating two paths from I' to (q, t) with the same labels, the transition from (p, s) to (q, t) with label q is not created if there exists already one from (p', s') to (q, t) for a state (p', s') that can be reached by a string also reaching (p, s) (condition of line 15). Note that if (p, s) is extracted from $\mathcal{Q}$ before a state (p', s') with $(p', s') R (p, s)$, then the transition from (p, s) to (q, t) is created first and the one from (p', s') to (q, t) not created. This is how the queue discipline directs the choice of the transitions created.

Lines 16-18 add (q, t) to Q' when it is not already in Q' and line 19 adds the new transition defined to E'. After creation of this transition, the destination state (q, t) is then put in relation with all states (q', t') reached by a transition labeled with $a \in \Sigma$ from a state (p', s') that is in relation with (p, s).

3.2. *Illustration*

Figure 3 illustrates the application of our disambiguation algorithm when applied to the simple automaton of Figure 2(a) and compares it with the construction of Schützenberger. As can be seen from the figures, the construction of Schützenberger results in more states (8 versus 6) and many more transitions than our disambiguation algorithm (21 versus 9). The construction of Schützenberger is based on the composition of the determinized automaton of Figure 2(b) with the automaton A of Figure 2(a), thus the states in its result are pairs made of a state of A and a subset constructed in the determinization of A.

The subsets constructed by our algorithm may not correspond to the determinization subsets. Observe for instance that states 1 or 2 are not included in the subset of $(0, \{0\})$ in the automaton of Figure 2(b) resulting from the application of our algorithm. This is because 0 does not share a common future with 1 or 2. Similarly, state 0 is not included in the subset $\{1, 2\}$ of state 1.

Some states created by the construction of Schützenberger or by our disambiguation algorithm may be non-coaccessible, that is, they may admit no transition to a final state because their output transitions were not constructed to avoid generating ambiguity. These states and the transitions leading to them can be removed in linear time using a standard trimming algorithm. In the case of the unambiguous automata of Figure 3, in each case, the state whose dashed transition is not constructed can be trimmed.

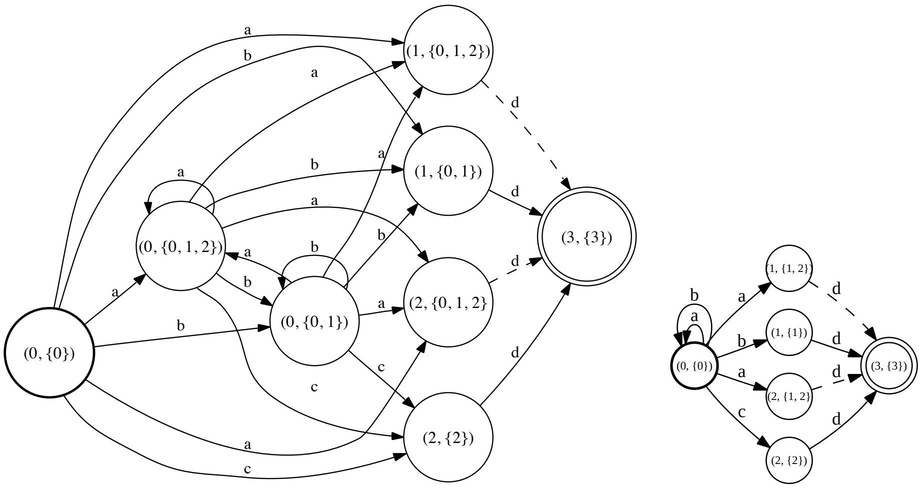
3.3. *Analysis*

The termination of the algorithm is guaranteed by the fact that the number of states and transitions created must be finite. This is because the number of possible subsets s of states of A is finite, thereby also the number of pairs (p, s) created by the algorithm where p is a state of A and s a subset. Also, the number of transitions created at a state (p, s) is at most equal to the number of states leaving p in A. In the worst case, the algorithm may create exponentially many subsets and thus the computational complexity of the algorithm is exponential. In many practical cases, however, this worst case behavior is not observed. In particular, the automaton returned by our disambiguation algorithm is substantially smaller than the one obtained by application of determinization.

The following proposition shows that the automaton returned by the algorithm is unambiguous. The proof is presented in Appendix A.

Proposition 2. *The automaton A' returned by algorithm* Disambiguation *run on input automaton A is unambiguous.*

The following lemma will be used to show the equivalence between the automaton returned by the algorithm and the input automaton. The proof is presented in



(a) (b)

Fig. 3. Illustration of the disambiguation algorithms. (a) Result of Schützenberger's construction applied to the automaton A of Figure 2(a). (b) Result of our disambiguation algorithm applied to the automaton A of Figure 2(a). In both cases, one of the two dashed transitions is disallowed by the algorithm.

Appendix B.

Lemma 3. *Let A' be the automaton returned by algorithm* Disambiguation *run on input automaton A, then $L(A') \subseteq L(A)$.*

Next, the lemma is used to prove the following proposition.

Proposition 4. *The automaton A' returned by algorithm* Disambiguation *run on input automaton A is equivalent to A.*

Proof. By construction, a path $((p_1, s_1), a_1, (p_2, s_2)) \cdots ((p_k, s_k), a_k, (p_{k+1}, s_{k+1}))$ is created in A' only if the path $(p_1, a_1, p_2) \cdots (p_k, a_k, p_{k+1})$ exists in A, and a state (p, s) is made final in A' only if p is final in A. Thus, if a string $x = a_1 \cdots a_k$ is accepted by A' it is also accepted by A, which shows that $L(A') \subseteq L(A)$. The reverse inclusion holds by Lemma 3. □

The following theorem follows directly by Propositions 2 and 4.

Theorem 5. *The automaton A' returned by algorithm* Disambiguation *run on input automaton A is an unambiguous automaton equivalent to A.*

Note that the states disallowed via the condition of our algorithm are the minimal ones that can be safely removed from the subsets to check the presence of ambiguities.

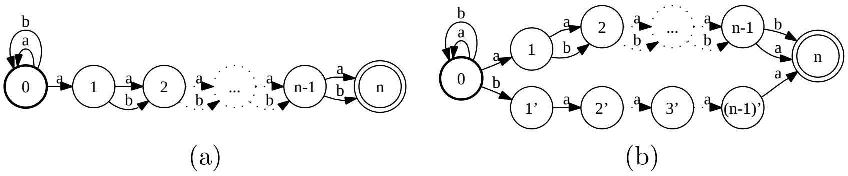


Fig. 4. Examples of automata A for which determinization returns an exponentially larger automaton while our algorithm returns A (for (a)) or an automaton whose size is linear in A (for (b)). (a) Automaton representing the regular expression $(a+b)^*a(a+b)^n$, whose minimal deterministic equivalent has size $\Omega(2^n)$. (b) Automaton representing the regular expression $(a+b)^*(a(a+b)^n+ba^n)$, whose determinization results in an automaton with $\Omega(2^n)$ states.

In the case of the automata of Figures 2-3, the construction of Schützenberger presents no clear benefit over that of determinization which it is based on: the construction is by definition more costly than determinization and in this case results in more states and transitions. We will show that this is a general fact. Let A be a trim finite automaton, let $\det(A)$ be the result of the determinization of A obtained by the standard subset construction, and $\text{schutz}(A)$ denote the result of Schützenberger's construction applied to A before deletion of any disallowed transition. In the composition of A with $\det(A)$, a subset $s=(q_1,\ldots,q_t)$ (state of $\det(A)$) is paired exactly once with each state q_i, $i\in[1,t]$, resulting in t states (q_i,s). Indeed, by definition of the subset construction, the states q_i can be reached by the same string which is also a string reaching s in $\det(A)$. Conversely, all the states constructed by composition are of this type. Thus, disregarding for now the transitions disallowed in Schützenberger's construction, the number of states of $\text{schutz}(A)$ is

$$|Q_{\text{schutz}(A)}| = \left[\sum_{s\in Q_{\det(A)}} |s|\right] \geq |Q_{\det(A)}|,$$

where $Q_{\det(A)}$ denotes the set of states of $\det(A)$ each representing a subset s of states of A, and $|s|$ the size of subset s. This proves that the number of states constructed is never less than the number of states of $\det(A)$. Furthermore, by definition of composition, each state (q_i,s) admits exactly as many outgoing transitions as the number of outgoing transitions of q_i in A since q_i is an element of the subset s. Thus, the number of transitions of the automaton constructed by Schützenberger's construction is

$$|E_{\text{schutz}(A)}| = \left[\sum_{s=(q_1,\ldots,q_t)\in Q_{\det(A)}} \sum_{i=1}^{t} |E(q_i)|\right] \geq |E_{\det(A)}|,$$

where $|E(q_i)|$ denotes the number of outgoing transitions of state q_i in A and $E_{\det(A)}$ the set of transitions in $\det(A)$. This shows that the result of the construction has also at least as many transitions as $\det(A)$. Thus, this shows that the inequality

$$|\operatorname{schutz}(A)| \geq |\det(A)|,$$

holds in general. Of course, since some transitions are disallowed, the actual number of transitions will be less. Additionally, this may yield some non-coaccessible states and transitions which can be removed by trimming the result. But, even after trimming, the size of the automaton cannot be less than that of $\det(A)$ since there needs to be at least one state (q_i, s) in the resulting trimmed machine and since each transition of $\det(A)$ must be paired with at least one transition leaving a state of this type.

In contrast, the size of the automaton created by our algorithm can be less, sometimes exponentially less, than that of automaton created by determinization. More generally, note that when the input automaton is unambiguous, the subsets created by our algorithm are reduced to singletons: by Proposition 1, a subset cannot contain two distinct states in that case. In such cases, our algorithm simply returns the same automaton A. The work done after computation of B is also linear in $|A|$. In contrast, the determinization of A may lead to a blow-up, even when the automaton is unambiguous. In particular, for the standard case of the non-deterministic automaton of Figure 4(a) representing the regular expression $(a + b)^*a(a + b)^n$, it is known that determinization creates $2^{n+1} - 1$ states. However, this automaton is unambiguous and our algorithm returns the same automaton unchanged. The automaton of Figure 4(b) is similar but is ambiguous. Nevertheless, it is not hard to see that again the size of the automaton returned by determinization is exponential and that that of the automaton output by our algorithm is only linear.

An alternative description of our algorithm can be used to compare the number of states created versus those created by the construction of Schützenberger. From the analysis that precedes, it should be clear that, when applied to an automaton A, the states created by our disambiguation can all be obtained straightforwardly from those created by the construction of Schützenberger as follows: to each state (q, s') created by the construction of Schützenberger corresponds a state (q, s) created by our algorithm where the subset s is derived from s' by removing the states that do not share a common future with q. This shows that our algorithm creates at most as many states as the construction of Schützenberger. Two distincts states (q, s'_1) and (q, s'_2) created by the construction of Schützenberger correspond to the same state (q, s) in our algorithm when both s'_1 and s'_2 reduce to s after removal of the states not sharing a common future with q. When this occurs for many states, our algorithm creates substantially less states. In particular, in the case of unambiguous automata, the subsets s in our algorithm are reduced to singletons $\{q\}$ since no two states share the same future.

4. Disambiguation of automata with ϵ-transitions

Our algorithm can also be extended to the case of automata with ϵ-transitions. We briefly describe that extension. Let A be an input automaton with ϵ-transitions.

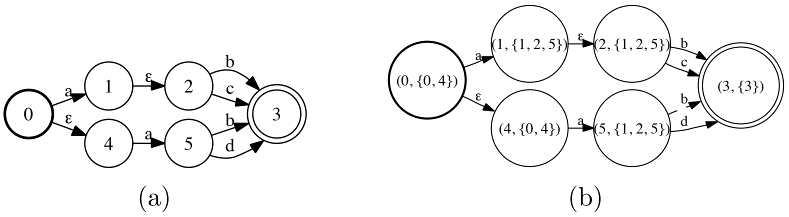


Fig. 5. (a) Automaton A with ϵ-transitions. (b) Unambiguous automaton equivalent to A returned by our disambiguation algorithm. The dashed transition is disallowed by the algorithm.

Here, the automaton B used to determine pairs of states sharing the same future is obtained similarly by computing the intersection $A \cap A$ by using an ϵ-filter [12] and by trimming the result by removing non-coaccessible states and transitions. For any set R of states of A, let $\epsilon[R]$ denote the ϵ-closure of R, that is the set of states reachable from states of R via paths labeled with ϵ.

To extend the algorithm to cover the case of automata with ϵ-transitions, it suffices to proceed as follows. The initial states are defined by the set of (i, s) with $i \in I$ and $s = \{q \in \epsilon[I] \colon (i, q) \in B\}$. At line 14, $\delta^*(s, a)$ is defined as the set of states reachable from s by reading a, including via ϵ-transitions. Finally, the relation R is extended to ϵ-transitions as follows: for each (p', s') such that $(p', s')\ R\ (p, s)$ and $((p, s), \epsilon, (q', t')) \in E'$, (p', s') is put in relation with (q', t'). Figure 5 illustrates the application of our algorithm in that case.

5. Disambiguation of finite-state transducers

In this section, we consider the problem of determining an unambiguous transducer equivalent to a given *functional finite-state transducer*, that is a finite-state transducer representing a (partial) rational function, or equivalently one associating at most one output string to any input string. The functionality of a finite-state transducer T can be tested efficiently from the transducer $T \circ T^{-1}$ as shown by [2].

Theorem 6 ([2]) *There exists an algorithm for testing the functionality of a finite-state transducer T with output alphabet Δ in time $O(|E|^2 + |\Delta|\,|Q|^2)$.*

One possible algorithm for finding an unambiguous transducer equivalent to a functional transducer is determinization [11], however, as discussed earlier, not all functional transducers admit an equivalent deterministic transducer. Figure 6(a) shows an example of such a functional transducer which in fact is unambiguous. A trim functional transducer is determinizable iff it admits the twins property [3].

We will describe instead a disambiguation algorithm does not require the property of determinizability. It is known that any functional transducer can be represented by an unambiguous transducer [9, 5]. For a functional transducer, by definition, two accepting paths with the same input label have the same output labels. Thus, for disambiguating a functional transducer, only input labels matter and our

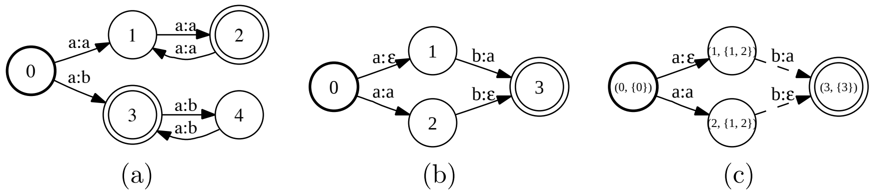


Fig. 6. (a) Unambiguous finite-state transducer admitting no sequential or deterministic equivalent. (b) Functional transducer T. (c) Disambiguated transducer equivalent to T returned by our algorithm. One of the two dashed transitions is disallowed by the algorithm.

automata disambiguation can be readily applied to create an unambiguous transducer equivalent to an input functional transducer. Our disambiguation algorithm gives a constructive proof of the existence of an equivalent unambiguous transducer for a rational function. The different possible *cross-sections* of the construction of [9] correspond to different orders in which transitions are visited and disallowed by our algorithm. Figure 6(b)-(c) illustrates the application of the algorithm in the case of a simple functional transducer.

As already pointed out, our algorithm compares favorably with the existing disambiguation algorithm for finite-state transducers of Schützenberger [17, 15]. That construction can be concisely described as follows. Let D be a deterministic automaton obtained by determinization of the input automaton A of the functional transducer T, that is the automaton obtained by removing the output labels of T. Then, the algorithm consists of composing D with T using the standard composition algorithm for finite-state transducers while disallowing finality of two composition states (p, s) and (q, s) with the same determinization subset s and distinct states p and q of T, and similarly disallowing all but one transition labeled with a from two states (p, s) and (q, s) to the same state, to avoid generating ambiguities. As can be seen from this description, the algorithm requires the determinization of A. This is implicit in the description of this construction in [14].

In contrast, our disambiguation algorithm does not require the determinization of A and as seen in the previous sections can return exponentially smaller automata than those returned by determinization is some cases. Consider for example the finite-state transducers defined as the automata of Figure 4 with each transition augmented with an output label identical to its output label. The construction of Schützenberger requires for those transducers the determinization of the input automata, thus its cost as well as the size of the result are exponential with respect to the size of the output as already discussed in Section 3. Unlike that construction, as in the automata case, our algorithm returns the same transducer or returns one whose size is only linear in that of the input.

The subsets defined by our disambiguation algorithm are never larger than those defined in the subset construction of determinization. As already discussed in the case of finite automata, this is because for a state (p, s) constructed in the algorithm,

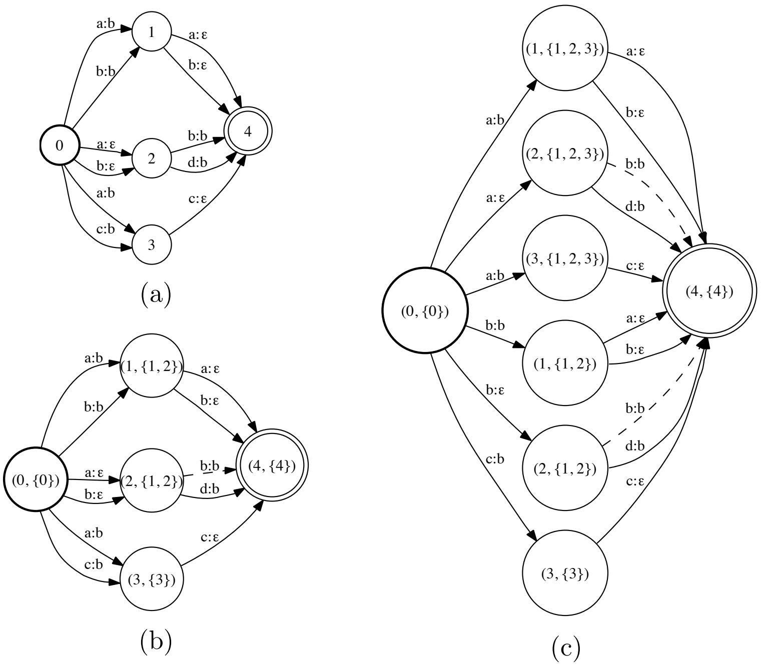


Fig. 7. Disambiguation of functional transducers. (a) Functional transducer T. (b) Unambiguous transducer equivalent to T returned by our algorithm. The dashed transitions are disallowed by the algorithm. (c) Unambiguous transducer returned by the disambiguation construction of Schützenberger .

only states sharing a common future with p are kept in the subset s. In addition to making the size of the subsets shorter, this also reduces the number of states created: two possible states (p, s') and $(p, s")$ in the construction of Schützenberger are reduced to the same (p, s) after removal from s' and $s"$ of the states not sharing a common future with p. This leads in many cases to transducers exponentially smaller than those generated by the construction of Schützenberger and similar improvements in time efficiency.

The observation just emphasized can be illustrated by the simple example of Figure 7. The transducer T of Figure 7(a) is functional but is not unambiguous. Figure 7(b) shows the result of our disambiguation algorithm which is an unambiguous transducer equivalent to T with the same number of states. In contrast, the transducer created by the construction of Schützenberger (Figure 7(c)) has several more states and transitions and some larger subsets.

6. Conclusion

We presented a new and often more efficient algorithm for the disambiguation of finite automata and functional transducers. This algorithm could be of great practical importance in a variety of applications including text and speech processing, bioinformatics, and in many other applications where they can be used to increase search efficiency. We have also designed a natural extension of these algorithms to some broad families of weighted automata and transducers defined over different semirings. We will present these extensions as well as their theoretical analysis in a future publication.

Acknowledgments

I thank Cyril Allauzen and Michael Riley for discussions about this work. This research was partly supported by a Google Research Award and by the NSF grant IIS-1117591.

References

[1] Jürgen Albert and Jarkko Kari. Digital image compression. In *Handbook of weighted automata*. Springer, 2009.

[2] Cyril Allauzen and Mehryar Mohri. Efficient algorithms for testing the twins property. *Journal of Automata, Languages and Combinatorics*, 8(2):117–144, 2003.

[3] Cyril Allauzen and Mehryar Mohri. Finitely subsequential transducers. *International Journal of Foundations of Computer Science*, 14(6):983–994, 2003.

[4] Cyril Allauzen, Mehryar Mohri, and Ashish Rastogi. General algorithms for testing the ambiguity of finite automata and the double-tape ambiguity of finite-state transducers. *Int. J. Found. Comput. Sci.*, 22(4):883–904, 2011.

[5] Jean Berstel. *Transductions and Context-Free Languages*. Teubner Studienbucher, 1979.

[6] Thomas M. Breuel. The OCRopus open source OCR system. In *Proceedings of IS&T/SPIE 20th Annual Symposium*, 2008.

[7] Christian Choffrut. *Contributions à l'étude de quelques familles remarquables de fonctions rationnelles*. PhD thesis, Université Paris 7, LITP: Paris, France, 1978.

[8] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[9] Samuel Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, 1974.

[10] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3), 1994.

[11] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

[12] Mehryar Mohri. Weighted automata algorithms. In *Handbook of Weighted Automata*, pages 213–254. Springer, 2009.

[13] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. Speech recognition with weighted finite-state transducers. In *Handbook on speech processing and speech communication*. Springer, 2008.

[14] Emmanuel Roche and Yves Schabes, editors. *Finite-State Language Processing*. MIT Press, 1997.

[15] Jacques Sakarovitch. A construction on finite automata that has remained hidden. *Theor. Comput. Sci.*, 204(1-2):205–231, 1998.

[16] Erik Meineche Schmidt. *Succinctness of Description of Context-Free, Regular and*

Unambiguous Languages. PhD thesis, Department of Computer Science, University of Aarhus, 1978.

[17] Marcel-Paul Schützenberger. Sur les relations rationnelles entre monoïdes libres. *Theor. Comput. Sci.*, 3(2):243–259, 1976.

Appendix A. Proof of Proposition 2

Lemma A.1. *Let (q, t) and (q', t') be two states constructed by algorithm* DISAMBIGUATION *run on input automaton A, then $(q, t)\ R\ (q', t')$ iff (q, t) and (q', t') are co-reachable.*

Proof. We will show by induction on the length of strings x that if two states (q, t) and (q', t') are both reachable by x, then $(p, s)\ R\ (q', t')$. The steps of lines 6-7 ensure that $(q, t)\ R\ (q', t')$ when both states are initial, that is, when they are reachable by ϵ. Assume that it holds for all strings x of length less than or equal to n. Let $x = x'a$ be a string of length $n + 1$ with $x' \in \Sigma^*$ and $a \in \Sigma$ and assume that (q, t) and (q', t') are both reachable by x. Then, there exists a state (p, s) reachable by x' and admitting a transition labeled with a leading to (q, t) and similarly a state (p', s') reachable by x' and admitting a transition labeled with a leading to (q', t'). Then, by the induction hypothesis, we have $(p, s)\ R\ (p', s')$, thus $(q, t)\ R\ (q', t')$ is guaranteed by execution of the steps of lines 20-21. This proves the implication corresponding to one side. The converse holds straightforwardly by construction (lines 6-7 and 20-21). □

Proof of Proposition 2. Let π_1 and π_2 be two paths in A' from I' to F' with the same label $x \in \Sigma^*$. If $x = \epsilon$, π_1 is a path from some initial state (i_1, s_1) to (i_1, s_1) and similarly π_2 a path from some initial state (i_2, s_2) to (i_2, s_2). All initial states are in relation (lines 6-7), therefore at most one can be made final (lines 11-12). This implies that $(i_1, s_1) = (i_2, s_2)$ and $\pi_1 = \pi_2$. Let (q_1, t_1) be the destination state of π_1 and (q_2, t_2) the destination state of π_2. Since (q_1, t_1) and (q_2, t_2) are both reachable by x, by Lemma A.1, we have $(q_1, t_1)\ R\ (q_2, t_2)$. Since no two distinct equivalent states can be made final (lines 11-12), we must have $(q_1, t_1) = (q_2, t_2)$.

If $x = \epsilon$, this implies that the two paths π_1 and π_2 coincide. If $x \neq \epsilon$, x can be written as $x = x'a$ with $x' \in \Sigma^*$ and $a \in \Sigma$ and π_1 and π_2 can be decomposed as $\pi_1 = \pi'_1 e_1$ and $\pi_2 = \pi'_2 e_2$ with e_1 and e_2 transitions labeled with a leading to (q_1, t_1). Let (p_1, s_1) be the destination state of π'_1 and (p_2, s_2) the destination state of π'_2. Since π'_1 and π'_2 are both labeled with x', by Lemma A.1, we have $(p_1, s_1)\ R\ (p'_1, s'_1)$. By the condition of line 15, if $(p_1, s_1) \neq (p'_1, s'_1)$, (p_1, s_1) and (p'_1, s'_1) cannot both admit a transition labeled with a and leading to the same state (q_1, t_1). Thus, we must have $(p_1, s_1) = (p'_1, s'_1)$. Proceeding in the same way with π'_1 and π'_2 and so on shows that the paths π_1 and π_2 coincide, which concludes the proof. □

Appendix B. Proof of Lemma 3

The proof of Lemma 3 relies on the following two lemmas.

Lemma B.2. *Let (p, s) be a state constructed by algorithm* DISAMBIGUATION *run on input automaton A. If (p, s) is reachable by the strings u and v in A', then the set of states reachable by u in A and sharing a common future with p coincides with the set of states reachable by v in A and sharing a common future with p .*

Proof. We show by recurrence on the length of u that if state (p, s) is reachable by u in A', then s is the set of states reachable by u and sharing a common future with p. This property holds straightforwardly for $u = \epsilon$ by the construction of lines 2-5. Assume now that it holds for all u of length less than or equal to n. Let $u = u'a$ with $u' \in \Sigma^*$ of length n and $a \in \Sigma$. If (p, s) is reachable by u, there must exist some state (p', s') reachable by u' and admitting a transition labeled with a leading to (p, s). By the induction hypothesis, s' is the set of states reachable by u' and sharing a common future with p'. By definition of s (line 14), $s = \{q \in \delta^*(s', a) \colon (q, p) \in B\}$, thus the states in s are all reachable by u and share a common future with p. Conversely, let q be a state reachable by u and sharing future with p. There is a transition labeled with a from some state q' reachable by u'. Since q' admits a transition to q labeled with a and p' admits a transition labeled with a to p, and p and q share a common future, p' and q' must also share a common future. By the induction hypothesis, s' is the set of states reachable by u' and sharing a common future with p', therefore q' is in s'. Since $q \in \delta^*(q', a)$ and q shares a common future with p, this implies that q is in s. This shows that the states in s are those reachable by u and sharing a common future with p. □

Lemma B.3. *Let A' be the automaton returned by algorithm* Disambiguation *run on input automaton A. Let q be a state reachable in A by string x. Then, there exists a state (q, t) in A' for some subset t such that (q, t) is reachable by x in A'.*

Proof. We will prove the property by induction on the length of x. The property straightforwardly holds for $x = \epsilon$ by the construction steps of lines 2-5. Assume now that it holds for all strings of length less than or equal to n and let $x = ua$ with u a string of length n and $a \in \Sigma$. If q is reachable by string x in A, then there exists a state p_0 in A reachable by u and admitting a transition labeled with a leading to q. By the induction hypothesis, there exists a state (p_0, s_0) in A' reachable by u. Now, the property clearly holds for (q, t_0) if the transition labeled with a leaving (p_0, s_0) is constructed at lines 15-19, with t_0 defined at line 14. Otherwise, by the test of line 15, there must exist in A' a distinct state (p_1, s'_0) admitting a transition labeled with a leading to (q, t_0) with $(p_1, s'_0)\ R\ (p_0, s_0)$. Note that we cannot have $p_1 = p_0$, since the same string cannot reach two distinct states (p_0, s_0) and (p_0, s_1). Now, since (p_1, s'_0) admits a transition labeled with a leading to (q, t_0), p_1 must admit a transition labeled with a and leading to q. Thus, p_1 and p_0 share a common future in A. Since $(p_1, s'_0)\ R\ (p_0, s_0)$, by Lemma A.1, they are reachable by a common string v. Thus, both u and v reach (p_0, s_0). By Lemma B.2, this implies that the set of states in A reachable by u and v and sharing a common future with p_0 are the same. Since p_1 and p_0 share a common future in A and v reaches both p_0 and p_1, u must also reach p_1 in A.

If u reaches (p_1, s'_0), then (q, t_0) can be reached by x since (p_1, s'_0) admits a transition labeled with a leading to (q, t). Otherwise, by the induction hypothesis, there must exist a distinct state (p_1, s_1) in A' reachable by u, with p_1 admitting a transition labeled with a to q. Reapplying the argument already presented for (p_0, s_0) to (p_1, s_1), either we find a path in A' labeled with x to a state (q, t_1), or there exists a state (p_2, s_2) in A' with the same property as (p_0, s_0) with p_2 distinct from p_1 and p_0. Since the number of distinct such states is finite, reiterating this process guarantees finding a path in A' labeled with x to a state (q, t_k) after some finite number of times k. Thus, the property holds in all cases. □

Proof of Lemma 3. The proof argument is similar to that of Lemma B.3. Let x be a string reaching a final state $q_0 \in F$ in A. By Lemma B.3, there exists a state (q_0, t_0) in A' reachable by x. If state (q_0, t_0) is made final (lines 11-12), this shows that x is accepted by A'. Otherwise, there must exist a final state (q_1, t'_0) with $(q_1, t'_0)\ R\ (q_0, t_0)$. Note that this implies that q_1 is final. Note also that we have $q_1 \neq q_0$ since two states (q_0, t_0) and (q_0, t'_0) cannot be co-reachable with $t'_0 \neq t_0$. Since $(q_1, t'_0)\ R\ (q_0, t_0)$, there exists a string x_1 reaching both states. Since (q_0, t_0) is reachable by both x and x_1, by Lemma B.2, the set of states in A reachable by x and sharing a common future with q_0 and those reachable by x_1 and sharing a common future with q_0 are the same. q_1 shares a common future with q_0 since both states are final and q_1 is reachable by x_1, therefore q_1 is reachable by x.

Now, if x reaches (q_1, t'_0), this shows that x is accepted by A'. Otherwise, by Lemma B.3, there exists a state (q_1, t_1) in A' reachable by x. We can reapply to (q_1, t_1) the same argument as for (q_0, t_0) since q_1 is a final state. Doing so, we either find a final state in A' reachable by x or a state (q_2, t_2) in A' with the same properties as (q_0, t_0) with q_0, q_1, and q_2 all distinct. Since the number of states of A' is finite, reiterating this process guarantees finding a final state reachable by x. This concludes the proof. □