Proceedings of HLT-NAACL 2003
Main Papers , pp. 64-71
Edmonton, May-June 2003

Simpler and More General Minimization for Weighted Finite-State Automata

Jason Eisner
Department of Computer Science
Johns Hopkins University
Baltimore, MD, USA 21218-2691
jason@cs.jhu.edu

Abstract

Previous work on minimizing weighted finite-state automata (including transducers) is limited to particular types of weights. We present efficient new minimization algorithms that apply much more generally, while being simpler and about as fast.

We also point out theoretical limits on minimization algorithms. We characterize the kind of "well-behaved" weight semirings where our methods work. Outside these semirings, minimization is not well-defined (in the sense of producing a unique minimal automaton), and even finding the minimum number of states is in general NP-complete and inapproximable.

1 Introduction

It is well known how to efficiently minimize a deterministic finite-state automaton (DFA), in the sense of constructing another DFA that recognizes the same language as the original but with as few states as possible (Aho et al., 1974). This DFA also has as few arcs as possible.

Minimization is useful for saving memory, as when building very large automata or deploying NLP systems on small hand-held devices. When automata are built up through complex regular expressions, the savings from minimization can be considerable, especially when applied at intermediate stages of the construction, since (for example) smaller automata can be intersected faster.

Recently the computational linguistics community has turned its attention to *weighted* automata that compute interesting *functions* of their input strings. A traditional automaton only returns an boolean from the set $K = \{true, false\}$, which indicates whether it has accepted the input. But a probabilistic automaton returns a probability in $K = [0, 1]$, or equivalently, a negated log-probability in $K = [0, \infty]$. A transducer returns an output string from $K = \Delta^*$ (for some alphabet Δ).

Celebrated algorithms by Mohri (1997; 2000) have recently made it possible to minimize deterministic automata whose weights (outputs) are log-probabilities or strings. These cases are of central interest in language and speech processing.

However, automata with other kinds of weights can also be defined. The general formulation of weighted automata (Berstel and Reutenauer, 1988) permits *any* weight set K, if appropriate operations $\oplus$ and $\otimes$ are provided for combining weights from the different arcs of the automaton. The triple $(K, \oplus, \otimes)$ is called a **weight semiring** and will be explained below. K-valued functions that can be computed by finite-state automata are called **rational functions**.

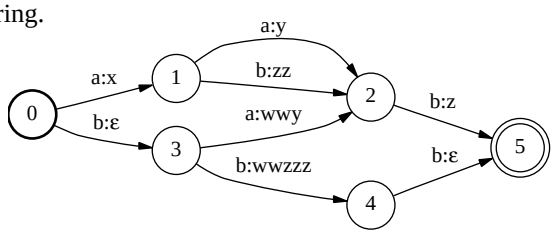
How does minimization generalize to arbitrary weight semirings? The question is of practical as well as theoretical interest. Some NLP automata use the *real semiring* $(\mathbb{R}, +, \times)$, or its log equivalent, to compute unnormalized probabilities or other scores outside the range $[0, 1]$ (Lafferty et al., 2001; Cortes et al., 2002). *Expectation semirings* (Eisner, 2002) are used to handle bookkeeping when training the parameters of a probabilistic transducer. A byproduct of this paper is a minimization algorithm that works fully with those semirings, a new result permitting more efficient automaton processing in those situations.

Surprisingly, we will see that minimization is not even well-defined for all weight semirings! We will then (nearly) characterize the semirings where it *is* well-defined, and give a recipe for constructing minimization algorithms similar to Mohri's in such semirings.

Finally, we follow this recipe to obtain a specific, simple and practical algorithm that works for all *division semirings*. All the cases above either fall within this framework or can be forced into it by adding multiplicative inverses to the semiring. The new algorithm provides arguably simpler minimization for the cases that Mohri has already treated, and also handles additional cases.

2 Weights and Minimization

We introduce weighted automata by example. The transducer below describes a partial function from strings to strings. It maps aab ↦ xyz and bab ↦ wwyz. Why? Since the transducer is deterministic, each input (such as aab) is accepted along at most one path; the corresponding output (such as xyz) is found by concatenating the output strings found along the path. ε denotes the empty string.



δ and σ standardly denote the automaton's **transition** and **output functions**: $\delta(3, \mathtt{a}) = 2$ is the state reached by the

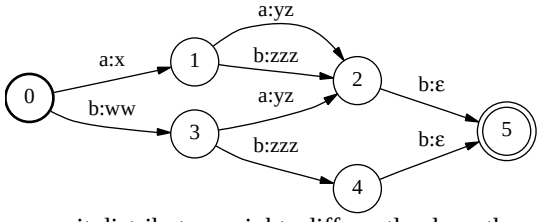
a arc from state 3, and $\sigma(3, \texttt{a}) = \texttt{wwy}$ is that arc's output.

In an automaton whose outputs (weights) were numbers rather than strings like wwy, concatenating them would not be sensible; instead we would want to add or multiply the weights along the path. In general $\otimes$ denotes the chosen operation for combining weights along a path.

The $\otimes$ operation need not be commutative—indeed concatenation is not—but it must be associative. K must contain (necessarily unique) weights, denoted $\underline{1}$ and $\underline{0}$, such that $\underline{1} \otimes k = k \otimes \underline{1} = k$ and $\underline{0} \otimes k = k \otimes \underline{0} = \underline{0}$ for all $k \in K$. An unaccepted input (e.g., aba) is assigned the output $\underline{0}$. When $\otimes$ is string concatenation, $\underline{1} = \varepsilon$, and $\underline{0}$ is a special object $\emptyset$ defined to satisfy the axioms.

If an input such as aa were accepted along multiple paths, we would have to use another operation $\oplus$ to combine those paths' weights into a single output for aa. But that cannot happen with the *deterministic* automata treated by this paper. So we omit discussion of the properties that $\oplus$ should have, and do not trouble to spell out its definition for the semirings $(K, \oplus, \otimes)$ discussed in this paper.[1] We are only concerned with the monoid $(K, \otimes)$.

The following automaton is equivalent to the previous one since it computes the same function:



However, it distributes weights differently along the arcs, and states 1 and 3 can now obviously be merged (as can 2 and 4, yielding the *minimal* equivalent automaton). Formally we know that states 1 and 3 are **equivalent** because $F_1 = F_3$, where F_q denotes the **suffix function** of state q—the function defined by the automaton if the start state is taken to be q rather than 0. (Thus, $F_3(\texttt{ab}) = \texttt{yz}$.) Equivalent states can safely be merged, by deleting one and rerouting its incoming arcs to the other.

We will follow Mohri's minimization strategy:

1. Turn the first automaton above into the second. This operation is called **pushing** (or **quasi-determinization**). Here, for instance, it "pushed ww back" through state 3.

2. **Merge** equivalent states of the second automaton, by applying ordinary *unweighted* DFA minimization (Aho et al., 1974, section 4.13) as if each weighted arc label such as a:yz were simply a letter in a large alphabet.

3. **Trim** the result, removing useless states and arcs that are not on any accepting path (defined as a path whose weight is non-$\underline{0}$ because it has no missing arcs and its last state is final).

[1] Though appropriate definitions do exist for our examples. For example, take the $\oplus$ of two strings to be the shorter of the two, breaking ties by a lexicographic ordering.

Mohri (2000) proves that this technique finds the minimal automaton, which he shows to be unique up to placement of weights along paths.[2]

We will only have to modify step 1, generalizing pushing to other semirings. Pushing makes heavy use of **left quotients**: we adopt the notation $k\backslash m$ for an element of K such that $k \otimes (k\backslash m) = m$. This differs from the notation $k^{-1} \otimes m$ (in which k^{-1} denotes an actual element of K) because $k\backslash m$ need not exist nor be unique. For example, $\texttt{ww}\backslash\texttt{wwzzz} = \texttt{zzz}$ (a fact used above) but $\texttt{wwy}\backslash\texttt{wwzzz}$ does not exist since wwzzz does not begin with wwy.

If F is a function, α is a string, and k is a weight, we use some natural notation for functions related to F:

$k \otimes F$: $(k \otimes F)(\gamma) \stackrel{\text{def}}{=} k \otimes (F(\gamma))$

$k\backslash F$: a function (if one exists) with $k \otimes (k\backslash F) = F$

$\alpha^{-1}F$: $(\alpha^{-1}F)(\gamma) \stackrel{\text{def}}{=} F(\alpha\gamma)$ (standard notation)

In effect, $k\backslash F$ and $\alpha^{-1}F$ drop output and input prefixes.

3 Pushing and Its Limitations

The intuition behind pushing is to canonicalize states' suffix functions. This increases the chance that two states will have the same suffix function. In the example of the previous section, we were able to replace F_3 with $\texttt{ww}\backslash F_3$ (pushing the ww backwards onto state 3's incoming arc), making it equal to F_1 so $\{1, 3\}$ could merge.

Since canonicalization was also performed at states 2 and 4, F_1 and F_3 ended up with identical representations: arc weights were distributed identically along corresponding paths from 1 and 3. Hence unweighted minimization could *discover* that $F_1 = F_3$ and merge $\{1, 3\}$.

Mohri's pushing strategy—we will see others—is always to extract some sort of "maximum left factor" from each suffix function F_q and push it backwards. That is, he expresses $F_q = k \otimes G$ for as "large" a $k \in K$ as possible—a maximal common prefix—then pushes factor k back out of the suffix function so that it is counted earlier on paths through q (i.e., *before* reaching q). q's suffix function now has canonical form G (i.e., $k\backslash F_q$).

How does Mohri's strategy reduce to practice? For transducers, where $(K, \otimes) = (\Delta^*, \text{concat})$, the maximum left factor of F_q is the longest common prefix of the strings in $\mathsf{range}(F_q)$.[3] Thus we had $\mathsf{range}(F_3) = \{\texttt{wwyz}, \texttt{wwzzz}\}$ above with longest common prefix ww. For the *tropical semiring* $(\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, +)$, where $k\backslash m = m - k$ is defined only if $k \leq m$, the maximum left factor k is the minimum of $\mathsf{range}(F_q)$.

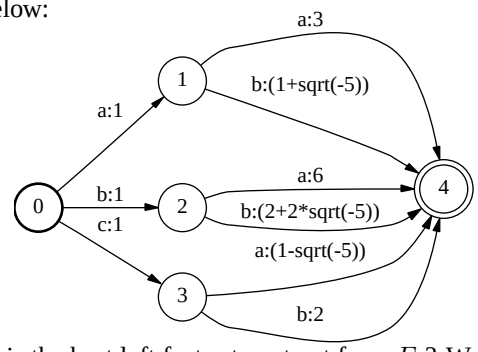
But "maximum left factor" is not an obvious notion for all semirings. If we extended the tropical semir-

[2] That is, any other solution is isomorphic to the one found here if output weights are ignored.

[3] In general we treat F_q as a *partial* function, so that $\mathsf{range}(F_q)$ excludes $\underline{0}$ (the weight of unaccepted strings). Left factors are unaffected, as *anything* can divide $\underline{0}$.

ing with negative numbers, or substituted the semiring $(\mathbb{R}_{\geq 0}, +, \times)$, keeping the usual definition of "maximum," then any function would have arbitrarily large left factors.

A more fundamentally problematic example is the semiring $\mathbb{Z}[\sqrt{-5}]$. It is defined as $(\{m+n\sqrt{-5} : m, n \in \mathbb{Z}\}, +, \times)$ where $\mathbb{Z}$ denotes the integers. It is a standard example of a commutative algebra in which factorization is not unique. For example, $6 = 2 \otimes 3 = (1+\sqrt{-5}) \otimes (1-\sqrt{-5})$ and these 4 factors cannot be factored further. This makes it impossible to canonicalize F_2 below:



What is the best left factor to extract from F_2? We could left-divide F_2 by either 2 or $1+\sqrt{-5}$. The former action allows us to merge $\{1,2\}$ and the latter to merge $\{2,3\}$; but we cannot have it both ways. So this automaton has no unique minimization! The minimum of 4 states is achieved by two distinct answers (contrast footnote 2).

It follows that *known minimization techniques will not work in general semirings*, as they assume state mergeability to be transitive.[4] In general the result of minimization is not even well-defined (i.e., unique).

Of course, given a deterministic automaton M, one may still seek an equivalent $\bar{M}$ with as few states as possible. But we will now see that even *finding the minimum number of states is NP-complete, and inapproximable*.

The NP-hardness proof [which may be skipped on a first reading] is by reduction from Minimum Clique Partition. Given a graph with vertex set $V = \{1, 2, \ldots n\}$ and edge set E, we wish to partition V into as few cliques as possible. ($S \subseteq V$ is a **clique** of the graph iff $ij \in E$ for all pairs $i, j \in S$.) Determining the minimum number of cliques is NP-complete and inapproximable: that is, unless P=NP, we cannot even find it within a factor of 2 or 3 or any other constant factor in polynomial time.[5]

Given such a graph, we reduce the clique problem to our problem. Consider the "bitwise boolean" semiring $(\{0,1\}^n, \text{OR}, \text{AND})$. Each weight k is a string of n bits, denoted $k_1, \ldots k_n$. For each $i \in V$, define $f^i, k^i, m^i \in K$ as follows: $f^i_j = 0$ iff $ij \in E$; $k^i_j = 1$ iff $i = j$; $m^i_j = 0$ iff either $ij \in E$ or $i = j$. Now consider the following automaton M over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c_1}, \ldots \mathtt{c_n}\}$. The states are $\{0, 1, \ldots n, n+1\}$; 0 is the initial state and $n+1$ is the only final state. For each $i \in V$, there is an arc $0 \xrightarrow{\mathtt{c_i}:1^n} i$ and arcs $i \xrightarrow{\mathtt{a}:k^i} (n+1)$ and $i \xrightarrow{\mathtt{b}:m^i} (n+1)$.

A minimum-state automaton equivalent to M *must* have a topology obtained by merging some states of V. Other topologies that could accept the same language $(\mathtt{c_1}|\mathtt{c_2}|\cdots|\mathtt{c_n})(\mathtt{a}|\mathtt{b})$ are clearly not minimal (they can be improved by merging final states or by trimming).

We claim that for $S \subseteq \{1, 2, \ldots n\}$, it is possible to merge all states in S into a single state (in the automaton) if and only if S is a clique (in the graph):

- If S is a clique, then define $k, m \in K$ by $k_i = 1$ iff $i \in S$, and $m_i = 1$ iff $i \notin S$. Observe that for every $i \in S$, we have $k^i = f^i \otimes k$, $m^i = f^i \otimes m$. So by pushing back a factor of f^i at each $i \in S$, one can make all $i \in S$ share a suffix function and then merge them.
- If S is not a clique, then choose $i, j \in S$ so that $ij \notin E$. Considering only bit i, there exists no bit pair $(k_i, m_i) \in \{0,1\}^2$ of which $(k^i_i, m^i_i) = (1, 0)$ and $(k^j_i, m^j_i) = (0, 1)$ are both left-multiples. So there can exist no weight pair (k, m) of which (k^i, m^i) and (k^j, m^j) are both left-multiples. It is therefore not possible to equalize the suffix functions F_i and F_j by left-dividing each of them.[6] i and j cannot be merged.

Thus, the partitions of V into cliques are identical to the partitions of V into sets of mergeable states, which are in 1-1 correspondence with the topologies of automata equivalent to M and derived from it by merging. There is an N-clique partition of V iff there is an $(N+2)$-state automaton. It follows that finding the *minimum* number of states is as hard, and as hard to approximate within a constant factor, as finding the minimum number of cliques.

4 When Is Minimization Unique?

The previous section demonstrated the existence of pathological weight semirings. We now partially characterize the "well-behaved" semirings $(K, \oplus, \otimes)$ in which all automata *do* have unique minimizations. Except when otherwise stated, *lowercase variables are weights* $\in K$ *and uppercase ones are* K*-valued rational functions*. [This section may be skipped, except the last paragraph.]

A crucial necessary condition is that $(K, \otimes)$ allow what we will call **greedy factorization**, meaning that given $f \otimes F = g \otimes G \neq \underline{0}$, it is always possible to express

[4] A further wrinkle lies in deciding what and how to push; in general semirings, it can be necessary to shift weights forward as well as backward along paths. Modify the example above by pushing a factor of 2 backwards through state 2. Making $F_2 = F_3$ in this modified example now requires pushing 2 forward and then $1+\sqrt{-5}$ backward through state 2.

[5] This problem is just the dual of Graph Coloring. For detailed approximability results see (Crescenzi and Kann, 1998).

[6] This argument only shows that pushing backward cannot give them the same suffix function. But pushing forward cannot help either, despite footnote 4, since 1^n on the arc to i has no right factors other than itself (the identity) to push forward.

$F = f' \otimes H$ and $G = g' \otimes H$. This condition holds for many practically useful semirings, commutative or otherwise. It says, roughly, that the order in which left factors are removed from a suffix function does not matter. We can reach the same canonical H regardless of whether we left-divide first by f or g.

Given a counterexample to this condition, one can construct an automaton with no unique minimization. Simply follow the plan of the $\mathbb{Z}[\sqrt{-5}]$ example, putting $F_1 = F$, $F_2 = f \otimes F = g \otimes G$, $F_3 = G$.[7] For example, in semiring $(K, \otimes) = (\{\mathsf{x}^n : n \neq 1\}, \text{concat})$, put $F_2 = \mathsf{x}^2 \otimes \{(\mathsf{a}, \mathsf{x}^3), (\mathsf{b}, \mathsf{x}^4)\} = \mathsf{x}^3 \otimes \{(\mathsf{a}, \mathsf{x}^2), (\mathsf{b}, \mathsf{x}^3)\}$.

Some *useful* semirings do fail the condition. One is the "bitwise boolean" semiring that checks a string's membership in *two* languages at once: $(K, \oplus, \otimes) = (\{00, 01, 10, 11\}, \text{OR}, \text{AND})$. (Let $F_2 = 01 \otimes \{(a, 11), (b, 00)\} = 01 \otimes \{(a, 01), (b, 10)\}$.) $\mathbb{R}^2$ under pointwise $\times$ (which computes a string's probability under *two* models) fails similarly. So does $(\text{sets}, \cap, \cup)$ (which collects features found along the accepting path).

We call H a **residue** of F iff $F = f' \otimes H$ for some f'. Write $F \simeq G$ iff F, G have a *common* residue. In these terms, $(K, \otimes)$ allows greedy factorization iff $F \simeq G$ when F, G are residues of the same nonzero function. More perspicuously, one can show that this holds iff $\simeq$ is an equivalence relation on nonzero, K-valued functions.

So in semirings where minimization is uniquely defined, $\simeq$ is necessarily an equivalence relation. Given an automaton M for function F, we may regard $\simeq$ as an equivalence relation on the states of a trimmed version of M:[8] $q \simeq r$ iff $F_q \simeq F_r$. Let $[r] = \{r_1, \ldots, r_m\}$ be the (finite) equivalence class of r: we can inductively find at least one function $F_{[r]}$ that is a common residue of $F_{r_1}, \ldots, F_{r_m}$. The idea behind minimization is to construct a machine $\bar{M}$ whose states correspond to these equivalence classes, and where each $[r]$ has suffix function $F_{[r]}$. The Appendix shows that $\bar{M}$ is then minimal. If M has an arc $q \xrightarrow{a:k} r$, $\bar{M}$ needs an arc $[q] \xrightarrow{a:k'} [r]$, where k' is such that $a^{-1}F_{[q]} = k' \otimes F_{[r]}$.

The main difficulty in completing the construction of $\bar{M}$ is to ensure each weight k' exists. That is, $F_{[r]}$ must be carefully chosen to be a residue not only of $F_{r_1}, \ldots, F_{r_m}$ (which ultimately does not matter, as long as $F_{[0]}$ is a residue of F_0, where 0 is the start state) but also of $a^{-1}F_{[q]}$. If M is cyclic, this imposes cyclic dependencies on the choices of the various $F_{[q]}$ and $F_{[r]}$ functions.

We have found no simple necessary and sufficient condition on $(K, \otimes)$ that guarantees a globally consistent set of choices to exist. However, we have given a useful necessary condition (greedy factorization), and we now give a useful sufficient condition. Say that H is a **minimum residue** of $G \neq 0$ if it is a residue of every residue of G. (If G has several minimum residues, they are all residues of one another.) If $(K, \otimes)$ is such that *every* G has a minimum residue—a strictly stronger condition than greedy factorization—then it can be shown that G has the same minimum residues as any $H \simeq G$. In such a $(K, \otimes)$, $\bar{M}$ can be constructed by choosing the suffix functions $F_{[r]}$ independently. Just let $F_{[r]} = F_{\{r_1, \ldots, r_m\}}$ be a minimum residue of F_{r_1}. Now consider again M's arc $q \xrightarrow{a:k} r$: since $a^{-1}F_{[q]} \simeq a^{-1}F_q \simeq F_r \simeq F_{r_1}$, we see $F_{[r]}$ is a (minimum) residue of $a^{-1}F_{[q]}$, so that a weight k' can be chosen for $[q] \xrightarrow{a:k'} [r]$.

A final step ensures that $\bar{M}$ defines the function F. To describe it, we must augment the formalism to allow an **initial weight** $\iota(0) \in K$, and a **final weight** $\phi(r) \in K$ for each final state r. The weight of an accepting path from the start state 0 to a final state r is now defined to be $\iota(0) \otimes$ (weights of arcs along the path) $\otimes \phi(r)$. In $\bar{M}$, we set $\iota([0])$ to some k such that $F_0 = k \otimes F_{[0]}$, and set $\phi([r]) = F_{[r]}(\varepsilon)$. The mathematical construction is done.

5 A Simple Minimization Recipe

We now give an effective algorithm for minimization in the semiring $(K, \otimes)$. The algorithmic recipe has one ingredient: along with $(K, \otimes)$, the user must give us a **left-factor functional** λ that can choose a left factor $\lambda(F)$ of any function F. Formally, if Σ is the input alphabet, then we require $\lambda : (\Sigma^* \to K) \to K$ to have the following properties for any rational $F : \Sigma^* \to K$ and any $k \in K$:

- **Shifting**: $\lambda(k \otimes F) = k \otimes \lambda(F)$.
- **Quotient**: $\lambda(F) \backslash \lambda(a^{-1}F)$ exists in K for any $a \in \Sigma$.
- **Final-quotient**: $\lambda(F) \backslash F(\varepsilon)$ exists in K.[9]

The algorithm generalizes Mohri's strategy as outlined in section 2. We just use λ to pick the left factors during pushing. The λ's used by Mohri for two semirings were mentioned in section 3. We will define another λ in section 6. Naturally, it can be shown that no λ can exist in a semiring that lacks greedy factorization, such as $\mathbb{Z}[\sqrt{-5}]$.

The 3 properties above are needed for the strategy to work. The strategy also requires $(K, \otimes)$ to be **left cancellative**, i.e., $k \otimes m = k \otimes m'$ implies $m = m'$ (if $k \neq \underline{0}$). In other words, left quotients by k are unique when they exist (except for $\underline{0} \backslash \underline{0}$). This relieves us from having to make arbitrary choices of weight during pushing. Incompatible choices might prevent arc labels from matching as desired during the merging step of section 2.

[7] Then factoring F_2 allows state 2 to merge with either 1 or 3; but all three states cannot merge, since any suffix function that could be shared by 1 and 3 could serve as H.

[8] Trimming ensures that suffix functions are nonzero.

[9] To show the final-quotient property given the other two, it suffices to show that $\lambda(G) \in K$ has a right inverse in K, where G is the function mapping ε to $\underline{1}$ and everything else to $\underline{0}$.

Given an input DFA. At each state q, simultaneously, we will push back $\lambda(F_q)$. This pushing construction is trivial once the $\lambda(F_q)$ values are computed. An arc $q \xrightarrow{a:k} r$ should have its weight changed from k to $\lambda(F_q)\backslash\lambda(a^{-1}F_q) = \lambda(F_q)\backslash\lambda(k \otimes F_r)$, which is well-defined (by the quotient property and left cancellativity)[10] and can be computed as $\lambda(F_q)\backslash(k \otimes \lambda(F_r))$ (by the shifting property). Thus a subpath $q \xrightarrow{a:k} r \xrightarrow{b:\ell} s$, with weight $k \otimes \ell$, will become $q \xrightarrow{a:k'} r \xrightarrow{b:\ell'} s$, with weight $k' \otimes \ell' = (\lambda(F_q)\backslash(k \otimes \lambda(F_r))) \otimes (\lambda(F_r)\backslash(\ell \otimes \lambda(F_s)))$. In this way the factor $\lambda(F_r)$ is removed from the start of all paths *from* r, and is pushed backwards through r onto the end of all paths *to* r. It is possible for this factor (or part of it) to travel back through multiple arcs and around cycles, since k' is found by removing a $\lambda(F_q)$ factor from all of $k \otimes \lambda(F_r)$ and not merely from k.

As it replaces the arc weights, pushing also replaces the initial weight $\iota(0)$ with $\iota(0) \otimes \lambda(F_0)$, and replaces each final weight $\phi(r)$ with $\lambda(F_r)\backslash\phi(r)$ (which is well-defined, by the final-quotient property). Altogether, pushing leaves path weights unchanged (by easy induction).[11]

After pushing, we finish with merging and trimming as in section 2. While merging via unweighted DFA minimization treats *arc* weights as part of the input symbols, what should it do with any initial and final weights? The start state's initial weight should be preserved. The merging algorithm can and should be initialized with a multi-way partition of states by final weight, instead of just a 2-way partition into final vs. non-final.[12]

The Appendix shows that this strategy indeed finds the unique minimal automaton.

It is worth clarifying how this section's effective algorithm implements the mathematical construction from the end of section 4. At each state q, pushing replaces the suffix function F_q with $\lambda(F_q)\backslash F_q$. The quotient properties of λ are designed to guarantee that this quotient is defined,[13] and the shifting property is designed to ensure that it is a minimum residue of F_q.[14] In short, if the conditions of this section are satisfied, so are the conditions of section 4, and the construction is the same.

The converse is true as well, at least for right cancellative semirings. If such a semiring satisfies the conditions of section 4 (every function has a minimum residue), then the requirements of this section can be met to obtain an effective algorithm: there exists a λ satisfying our three properties,[15] and the semiring is left cancellative.[16]

6 Minimization in Division Semirings

For the most important idea of this paper, we turn to a common special case. Suppose the semiring $(K, \oplus, \otimes)$ defines $k\backslash m$ for all $m, k \neq \underline{0} \in K$. Equivalently,[17] suppose every $k \neq \underline{0} \in K$ has a unique two-sided inverse $k^{-1} \in K$. Useful cases of such **division semirings** include the real semiring $(\mathbb{R}, +, \times)$, the tropical semiring extended with negative numbers $(\mathbb{R} \cup \{\infty\}, \min, +)$, and expectation semirings (Eisner, 2002). Minimization has not previously been available in these.

We propose a new left-factor functional that is fast to compute and works in *arbitrary* division semirings. We avoid the temptation to define $\lambda(F)$ as $\bigoplus \mathsf{range}(F)$: this definition has the right properties, but in some semirings including $(\mathbb{R}_{\geq 0}, +, \times)$ the infinite summation is quite expensive to compute and may even diverge. Instead (unlike Mohri) we will permit our $\lambda(F)$ to depend on more than just $\mathsf{range}(F)$.

Order the space of input strings Σ^* by length, breaking ties lexicographically. For example, $\varepsilon < bb < aab < aba < abb$. Now define

[10] Except in the case $\underline{0}\backslash\underline{0}$, which is not uniquely defined. This arises only if $F_q = 0$, i.e., q is a dead state that will be trimmed later, so any value will do for $\underline{0}\backslash\underline{0}$: arcs from q are irrelevant.

[11] One may prefer a formalism without initial or final weights. If the original automaton is free of final weights (other than $\underline{1}$), so is the pushed automaton—provided that $\lambda(F) = \underline{1}$ whenever $F(\varepsilon) = \underline{1}$, as is true for all λ's in this paper. Initial weights can be eliminated at the cost of duplicating state 0 (details omitted).

[12] Alternatively, Mohri (2000, §4.5) explains how to temporarily eliminate final weights before the merging step.

[13] That is, $\lambda(F_q)\backslash F_q(\gamma)$ exists for each $\gamma \in \Sigma^*$. One may show by induction on $|\gamma|$ that the left quotients $\lambda(F)\backslash F(\gamma)$ exist for *all* F. When $|\gamma| = 0$ this is the final-quotient property. For $|\gamma| > 0$ we can write γ as $a\gamma'$, and then $\lambda(F)\backslash F(\gamma) = \lambda(F)\backslash F(a\gamma') = \lambda(F)\backslash(a^{-1}F)(\gamma') = (\lambda(F)\backslash\lambda(a^{-1}F)) \otimes (\lambda(a^{-1}F)\backslash(a^{-1}F)(\gamma'))$, where the first factor exists by the quotient property and the second factor exists by inductive hypothesis.

[14] Suppose X is any residue of F_q, i.e., we can write $F_q = x \otimes X$. Then we can rewrite the identity $F_q = \lambda(F_q) \otimes (\lambda(F_q)\backslash F_q)$, using the shifting property, as $x \otimes X = x \otimes \lambda(X) \otimes (\lambda(F_q)\backslash F_q)$. As we have separately required the semiring to be left cancellative, this implies that $X = \lambda(X) \otimes (\lambda(F_q)\backslash F_q)$. So $(\lambda(F_q)\backslash F_q)$ is a residue of any residue X of F_q, as claimed.

[15] Define $\lambda(\underline{0}) = \underline{0}$. From each equivalence class of nonzero functions under $\simeq$, pick a single minimum residue (axiom of choice). Given F, let $[F]$ denote the minimum residue from its class. Observe that $F = f \otimes [F]$ for some f; right cancellativity implies f is unique. So define $\lambda(F) = f$. *Shifting property:* $\lambda(k \otimes F) = \lambda(k \otimes f \otimes [F]) = k \otimes f = k \otimes \lambda(f \otimes [F]) = k \otimes \lambda(F)$. *Quotient property:* $\lambda(a^{-1}F) \otimes [a^{-1}F] = a^{-1}F = a^{-1}(\lambda(F) \otimes [F]) = \lambda(F) \otimes a^{-1}[F] = \lambda(F) \otimes \lambda(a^{-1}[F]) \otimes [a^{-1}[F]] = \lambda(F) \otimes \lambda(a^{-1}[F]) \otimes [a^{-1}F]$ (the last step since $a^{-1}[F] \simeq a^{-1}F$). Applying right cancellativity, $\lambda(a^{-1}F) = \lambda(F) \otimes \lambda(a^{-1}[F])$, showing that $\lambda(F)\backslash\lambda(a^{-1}F)$ exists. *Final-quotient property:* Quotient exists since $F(\varepsilon) = \lambda(F) \otimes [F](\varepsilon)$.

[16] Let $\langle x, y \rangle$ denote the function mapping `a` to x, `b` to y, and everything else to $\underline{0}$. Given $km = km'$, we have $k \otimes \langle m, 1 \rangle = k \otimes \langle m', 1 \rangle$. Since the minimum residue property implies greedy factorization, we can write $\langle m, 1 \rangle = f \otimes \langle a, b \rangle$, $\langle m', 1 \rangle = g \otimes \langle a, b \rangle$. Then $f \otimes b = g \otimes b$, so by right cancellativity $f = g$, whence $m = f \otimes a = g \otimes a = m'$.

[17] The equivalence is a standard exercise, though not obvious.

$$\lambda(F) \stackrel{\text{def}}{=} \begin{cases} F(\min \textsf{support}(F)) \in K & \text{if } F \neq \underline{0} \\ \underline{0} & \text{if } F = \underline{0} \end{cases}$$

where $\textsf{support}(F)$ denotes the set of input strings to which F assigns a non-$\underline{0}$ weight. This λ clearly has the shifting property needed by section 5. The quotient and final-quotient properties come for free because we are in a division semiring and because $\lambda(F) = \underline{0}$ iff $F = \underline{0}$.

Under this definition, what is $\lambda(F_q)$ for a suffix function F_q? Consider all paths of nonzero weight[18] from state q to a final state. If none exist, $\lambda(F_q) = 0$. Otherwise, $\min \textsf{support}(F_q)$ is the input string on the shortest such path, breaking ties lexicographically.[19] $\lambda(F_q)$ is simply *the weight of that shortest path*.

To push, we must compute $\lambda(F_q)$ for each state q. This is easy because $\lambda(F_q)$ is the weight of a single, minimum-length and hence acyclic path from q. (Previous methods combined the weights of *all* paths from q, even if infinitely many.) It also helps that the left factors at different states are related: if the minimum path from q begins with a weight-k arc to r, then it continues along the minimum path from r, so $\lambda(F_q) = k \otimes \lambda(F_r)$.

Below is a trivial linear-time algorithm for computing $\lambda(F_q)$ at every q. Each state and arc is considered once in a breadth-first search back from the final states. *len*(q) and *first*(q) store the string length and first letter of a running minimum of $\textsf{support}(F_q) \in \Sigma^*$.

```
1.  foreach state q
2.      if q is final then
3.          len(q) := 0                    (* min support(F_q) is ε for final q *)
4.          λ(F_q) := φ(q)                 (* F_q(ε) is just the final weight, φ(q) *)
5.          enqueue q on a FIFO queue
6.      else
7.          len(q) := ∞                    (* not yet discovered *)
8.          λ(F_q) := 0                    (* assume F_q = 0 until we discover q *)
9.  until the FIFO queue is empty
10.     dequeue a state r
11.     foreach arc q --a:k--> r entering r such that k ≠ 0
12.         if len(q) = ∞ then enqueue q   (* breadth-first search *)
13.         if len(q) = ∞ or (len(q) = len(r) + 1
                                and a < first(q)) then
14.             first(q) := a              (* reduce min support(F_q) *)
15.             len(q) := len(r) + 1
16.             λ(F_q) := k ⊗ λ(F_r)
```

The runtime is $O(|\text{states}| + t \cdot |\text{arcs}|)$ if $\otimes$ has runtime t. If $\otimes$ is slow, this can be reduced to $O(t \cdot |\text{states}| + |\text{arcs}|)$ by removing line 16 and waiting until the end, when the minimum path from each non-final state q is fully known, to compute the weight $\lambda(F_q)$ of that path. Simply finish up by calling FIND-λ on each state q:

```
FIND-λ(state q):
1.  if λ(F_q) = 0 and len(q) < ∞ then
2.      λ(F_q) := σ(q, first(q)) ⊗ FIND-λ(δ(q, first(q)))
3.  return λ(F_q)
```

[18] In a division semiring, these are paths free of $\underline{0}$-weight arcs.

[19] The min exists since $<$ is a well-ordering. In a purely lexicographic ordering, $a^*b \subseteq \Sigma^*$ would have no min.

After thus computing $\lambda(F_q)$, we simply proceed with pushing, merging, and trimming as in section 5.[20] Pushing runs in time $O(t \cdot |\text{arcs}|)$ and trimming in $O(|\text{states}| + |\text{arcs}|)$. Merging is worse, with time $O(|\text{arcs}| \log |\text{states}|)$.

7 A Bonus: Non-Division Semirings

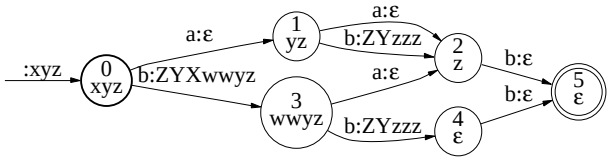
The trouble with $\mathbb{Z}[\sqrt{-5}]$ was that it "lacked" needed quotients. The example on p. 3 can easily be minimized (down to 3 states) if we regard it instead as defined over $(\mathbb{C}, +, \times)$—letting us use *any* weights in $\mathbb{C}$. Simply use section 6's algorithm.

This new change-of-semiring trick can be used for other non-division semirings as well. One can *extend* the original weight semiring $(K, \oplus, \otimes)$ to a division semiring by adding $\otimes$-inverses.[21]

In this way, the tropical semiring $(\mathbb{R}_{\geq 0} \cup \{\infty\}, \min, +)$ can be augmented with the *negative* reals to obtain $(\mathbb{R} \cup \{\infty\}, \min, +)$. And the transducer semiring $(\Delta^* \cup \{\emptyset\}, \min, \text{concat})$[22] can be augmented by extending the alphabet $\Delta = \{\mathtt{x}, \mathtt{y}, \ldots\}$ with **inverse letters** $\{\mathtt{x}^{-1}, \mathtt{y}^{-1}, \ldots\}$.

The minimized DFA we obtain may have "weird" arc weights drawn from the extended semiring. But the arc weights combine along paths to produce the original automaton's outputs, which fall in the original semiring.

Let us apply this trick to the example of section 2, yielding the following pushed automaton in which $F_1 = F_3$ as desired. ($\mathtt{x}^{-1}, \mathtt{y}^{-1}, \ldots$ are written as $\mathtt{X}, \mathtt{Y}, \ldots$, and $\lambda(F_q)$ is displayed at each q.)



For example, the $\mathtt{z}^{-1}\mathtt{y}^{-1}\mathtt{zzz}$ output on the $3 \rightarrow 4$ arc was computed as $\lambda(F_3)^{-1} \otimes \mathtt{wwzzz} \otimes \lambda(F_4) = (\mathtt{wwyz})^{-1} \otimes \mathtt{wwzzz} \otimes \varepsilon = \mathtt{z}^{-1}\mathtt{y}^{-1}\mathtt{w}^{-1}\mathtt{w}^{-1}\mathtt{wwzzz}$.

This trick yields new algorithms for the tropical semiring and sequential transducers, which is interesting and perhaps worthwhile. How do they compare with previous work?

Over the tropical semiring, our linear-time pushing algorithm is simpler than (Mohri, 1997), and faster by a

[20] It is also permissible to trim the input automaton at the start, or right after computing λ (note that $\lambda(F_q) = 0$ iff we should trim q). This simplifies pushing and merging. No trimming is then needed at the end, except to remove the one dead state that the merging step may have added to complete the automaton.

[21] This is often possible but not always; the semiring must be cancellative, and there are other conditions. Even disregarding $\oplus$ because we are minimizing a deterministic automaton, it is not simple to characterize when the monoid $(K, \otimes)$ can be embedded into a group (Clifford and Preston, 1967, chapter 12).

[22] Where min can be defined as in section 6 and footnote 1.

log factor, because it does not require a priority queue. (Though this does not help the overall complexity of minimization, which is dominated by the merging step.) We also have no need to implement faster algorithms for special cases, as Mohri proposes, because our basic algorithm is already linear. Finally, our algorithm generalizes better, as it can handle negative weight cycles in the input. These are useful in (e.g.) conditional random fields.

On the other hand, Mohri's algorithm guarantees a potentially useful property that we do not: that the weight of the prefix path reading $\alpha \in \Sigma^*$ is the minimum weight of all paths with prefix α. Commonly this approximates $-\log(p(\text{most probable string with prefix } \alpha))$, perhaps a useful value to look up for pruning.

As for transducers, how does our minimization algorithm (above) compare with previous ones? Following earlier work by Choffrut and others, Mohri (2000) defines $\lambda(F_q)$ as the longest common prefix of $\mathsf{range}(F_q)$. He constrains these values with a set of simultaneous equations, and solves them by repeated changes of variable using a complex relaxation algorithm. His implementation uses various techniques (including a trie and a graph decomposition) to make pushing run in time $O(|\text{states}| + |\text{arcs}| \cdot \max_q |\lambda(F_q)|)$.[23] Breslauer (1996) gives a different computation of the same result.

To implement our simpler algorithm, we represent strings in Δ^* as pointers into a global trie that extends upon lookup. The strings are actually stored reversed in the trie so that it is fast to add and remove short prefixes. Over the extended alphabet, we use the pointer pair (k, m) to represent the string $k^{-1}m$ where $k, m \in \Delta^*$ have no common prefix. Such pointer pairs can be equality-tested in $O(1)$ time during merging. For $k, m \in \Delta^*$, $k \otimes m$ is computed in time $O(|k|)$, and $k \backslash m$ in time $O(|\textsc{lcp}(k, m)|)$ or more loosely $O(|k|)$ (where LCP = longest common prefix).

The total time to compute our $\lambda(F_q)$ values is therefore $O(|\text{states}| + t \cdot |\text{arcs}|)$, where t is the maximum length of any arc's weight. For each arc we then compute a new weight as a left-quotient by a λ value. So our total runtime for pushing is $O(|\text{states}| + |\text{arcs}| \cdot \max_q |\lambda(F_q)|)$. This may appear identical to Mohri's runtime, but in fact our $|\lambda(F_q)| \geq$ Mohri's, though the two definitions share a worst case of $t \cdot |\text{states}|$.[24]

Inverse letters must be eliminated from the minimized transducer if one wishes to pass it to any specialized algorithms (composition, inversion) that assume weights in Δ^*. Fortunately this is not hard. If state q of the result was formed by merging states $q_1, \ldots q_j$, define $\rho(q) = \textsc{lcs}\{\lambda(F_{q_i}) : i = 1, \ldots j\} \in \Delta^*$ (where LCS = longest common suffix). Now push the minimized transducer using $\rho(q)^{-1}$ in place of $\lambda(F_q)$ for all q. This corrects for "overpushing": any letters $\rho(q)$ that were unnecessarily pushed back before minimization are pushed forward again, cancelling the inverse letters. In our running example, state 0 will push $(\mathtt{xyz})^{-1}$ back and the merged state {1,3} will push $(\mathtt{yz})^{-1}$ back. This is equivalent to pushing $\rho(0) = \mathtt{xyz}$ forward through state 0 and the $\mathtt{yz}$ part of it forward through {1,3}, canceling the $\mathtt{z}^{-1}\mathtt{y}^{-1}$ at the start of one of the next arcs.

We must show that the resulting labels really are free of inverse letters. Their values are as if the original pushing had pushed back not $\lambda(F_{q_i}) \in \Delta^*$ but only its shorter prefix $\hat{\lambda}(q_i) \stackrel{\text{def}}{=} \lambda(F_{q_i})/\rho(q_i) \in \Delta^*$ (note the *right* quotient). In other words, an arc from q_i to $r_{i'}$ with weight $k \in \Delta^*$ was reweighted as $\hat{\lambda}(q_i)\backslash(k \otimes \hat{\lambda}(r_{i'}))$. Any inverse letters in such new weights clearly fall at the left. So suppose the new weight on the arc from q to r begins with an inverse letter z^{-1}. Then $\hat{\lambda}(q_i)$ must have ended with z for each $i = 1, \ldots j$. But then $\rho(q_i)$ was not the longest common suffix: $z\rho(q_i)$ is a longer one, a contradiction (Q.E.D.).

Negative weights can be similarly eliminated after minimization over the tropical semiring, if desired, by substituting $\min$ for LCS.

The optional elimination of inverse letters or negative weights does not affect the asymptotic runtime. A caveat here is that the resulting automaton no longer has a canonical form. Consider a straight-line automaton: pushing yields a canonical form as always, but inverse-letter elimination completely undoes pushing ($\hat{\lambda}(q_i) = \varepsilon$). This is not an issue in Mohri's approach.

8 Conclusion and Final Remarks

We have characterized the semirings over which weighted deterministic automata can be minimized (section 4), and shown how to perform such minimization in both general and specific cases (sections 5, 6, 7). Our technique for division semirings and their subsemirings pushes back, at each state q, the output of a *single*, easily found, shortest accepting path from q. This is simpler and more general than previous approaches that aggregate *all* accepting paths from q.

Our new algorithm (section 6) is most important for previously unminimizable, practically needed division semirings: **real** (e.g., for probabilities), **expectation** (for learning (Eisner, 2002)), and **additive with negative weights** (for conditional random fields (Lafferty et al., 2001)). It can also be used in non-division semirings, as for transducers. It is unpatented, easy to implement,

[23] We define $|\varepsilon| = 1$ to simplify the $O(\cdots)$ expressions.

[24] The $|\lambda(F_q)|$ term contributed by a given arc from q is a bound on the length of the LCP of the outputs of certain paths from q. Mohri uses all paths from q and we use just two, so our LCP is sometimes longer. However, both LCPs probably tend to be short in practice, especially if one bypasses $\textsc{lcp}(k, k)$ with special handling for $k \backslash k = \varepsilon$.

comparable or faster in asymptotic runtime, and perhaps faster in practice (especially for the tropical semiring, where it seems preferable in most respects).

Our approach applies also to $\mathbb{R}$-weighted sequential transducers as in (Cortes et al., 2002). Such automata can be regarded as weighted by the product semiring $(\mathbb{R} \times \Delta^*, (+, \min), (\times, \text{concat}))$. Equivalently, one can push the numeric and string components independently.

Our new pushing algorithm enables not only minimization but also equivalence-testing in more weight semirings. Equivalence is efficiently tested by pushing the (deterministic) automata to canonicalize their arc labels and then testing unweighted equivalence (Mohri, 1997).

References

A. V. Aho, J. E. Hopcroft, and J. D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

Jean Berstel and Christophe Reutenauer. 1988. *Rational Series and their Languages*. Springer-Verlag.

Dany Breslauer. 1996. The suffix tree of a tree and minimizing sequential transducers. *Lecture Notes in Computer Science*, 1075.

A. H. Clifford and G. B. Preston. 1967. *The Algebraic Theory of Semigroups*.

Corinna Cortes, Patrick Haffner, and Mehryar Mohri. 2002. Rational kernels. In *Proceedings of NIPS*, December.

Pierluigi Crescenzi and Viggo Kann. 1998. How to find the best approximation results—a follow-up to Garey and Johnson. *ACM SIGACT News*, 29(4):90–97, December.

Jason Eisner. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proc. of ACL*, Philadelphia, July.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*.

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

Mehryar Mohri. 2000. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 324:177–201.

Appendix: Remaining Proofs

Let M be an automaton to minimize and $F : \Sigma^* \to K$ be the function it defines. We assume $(K, \otimes)$ allows greedy factorization, so $\simeq$ is an equivalence relation on nonzero functions. We first prove that $\bar{M}$ with the properties of section 4 is the minimal automaton computing F. We will then show, following Mohri, that the algorithm of section 5 finds such an $\bar{M}$. (Section 6 is a special case of section 5.)

We chose in advance a desired suffix function $F_{[r]}$ for each state $[r]$ of $\bar{M}$, and used these to determine the weights of $\bar{M}$. To show that the weights were determined correctly, let $\tilde{F}_{[r]}$ be the actual suffix function of $[r]$. Claim that for all α and r, $\tilde{F}_{[r]}(\alpha) = F_{[r]}(\alpha)$. This is easily proved by induction on $|\alpha|$. Our choice of initial weight then ensures that $\bar{M}$ computes F.

We must now prove minimality. For $\alpha, \beta \in \Sigma^*$, say $\alpha \overset{F}{\sim} \beta$ iff $\alpha^{-1}F \simeq \beta^{-1}F$. Note that $\overset{F}{\sim}$ is an equivalence relation on $D \overset{\text{def}}{=} \{\alpha \in \Sigma^* : \alpha^{-1}F \neq \underline{0}\}$.[25]

Let M' be any automaton that computes F. For $\alpha, \beta \in D$, we say $\alpha \overset{M'}{\sim} \beta$ iff $\delta_{M'}(0, \alpha) = \delta_{M'<}(0, \beta)$, i.e., the prefixes α and β lead from the start state 0 to the same state q in M'. If $\alpha \overset{M'}{\sim} \beta$, then $\alpha \overset{F}{\sim} \beta$, since $\alpha^{-1}F = \sigma(0, \alpha) \otimes F_q \simeq \sigma(0, \beta) \otimes F_q = \beta^{-1}F$.

If $\alpha \overset{F}{\sim} \beta$, then $\alpha^{-1}F \simeq \beta^{-1}F$, so $F_{\delta_M(0,\alpha)} \simeq \alpha^{-1}F \simeq \beta^{-1}F \simeq F_{\delta_M(0,\beta)}$, so $\delta_M(0, \alpha) \simeq \delta_M(0, \beta)$, so $\alpha \overset{\bar{M}}{\sim} \beta$ by construction of $\bar{M}$.

In short, $\alpha \overset{M'}{\sim} \beta \Rightarrow \alpha \overset{F}{\sim} \beta \Rightarrow \alpha \overset{\bar{M}}{\sim} \beta$. So each of the three partitions of D into equivalence classes is a refinement of the next. Hence $n_{M'} \geq n_F \geq n_{\bar{M}}$, where these are the respective numbers of equivalence classes.

Since $\overset{M'}{\sim}$ has one equivalence class per useful state of M' (as defined in section 2), $n_{M'}$ is the number of states in a trimmed version of M'. Similarly $n_{\bar{M}}$ is the number of states of $\bar{M}$ (after trimming). Since M' was arbitrary, $\bar{M}$ is minimal.

Uniqueness: If M' has the same number of states as $\bar{M}$, then the two partitions must be equal. So two prefixes reach the same state in M' iff they do so in $\bar{M}$. This gives a δ-preserving isomorphism between M' and $\bar{M}$. It follows that the minimal machine is unique, except for the distribution of output labels along paths (which may depend on arbitrary choices of residues $F_{[r]}$).

Now we turn to section 5's effective construction, using λ, of a pushed machine $\hat{M}$ and a merged version $\bar{M}$. The proof of minimality is essentially the same as in (Mohri, 2000). We know that $\bar{M}$ computes the same function as M (since pushing, merging, and trimming preserve this). So it suffices to show $\alpha \overset{F}{\sim} \beta \Rightarrow \alpha \overset{\bar{M}}{\sim} \beta$. The above proof of minimality will then go through as before.

M and $\hat{M}$ have the same states and transition function δ; denote their emission functions by σ and $\hat{\sigma}$. F_q refers to suffix functions in M. Given $\alpha \overset{F}{\sim} \beta$ (so $\alpha, \beta \in D$), use the definition of $\overset{F}{\sim}$ to write $\alpha^{-1}F = k_\alpha \otimes F'$ and $\beta^{-1}F = k_\beta \otimes F'$. Let $q = \delta(0, \alpha)$, $r = \delta(0, \beta), k = \sigma(0, \alpha)$. For any $a \in \Sigma$, write $\hat{\sigma}(q, a) = \lambda(F_q) \backslash \lambda(a^{-1}F_q) = (k \otimes \lambda(F_q)) \backslash (k \otimes \lambda(a^{-1}F_q)) = \lambda(k \otimes F_q) \backslash \lambda(k \otimes a^{-1}F_q) = \lambda(\alpha^{-1}F) \backslash \lambda(a^{-1}(\alpha^{-1}F)) = \lambda(k_\alpha \otimes F') \backslash \lambda(a^{-1}(k_\alpha \otimes F')) = \lambda(F') \backslash \lambda(a^{-1}F')$. By symmetry, $\hat{\sigma}(r, a) = \lambda(F') \backslash \lambda(a^{-1}F')$ as well. Thanks to left cancellativity, left quotients are unique, so $\hat{\sigma}(q, a) = \hat{\sigma}(r, a)$.[26]

So $\alpha \overset{F}{\sim} \beta \Rightarrow$ corresponding arcs from q and r in $\hat{M}$ output identical weights. Since $\alpha a \overset{F}{\sim} \beta a$ as well, the same holds at $\delta(q, a)$ and $\delta(r, a)$. So by induction, regarding $\hat{M}$ as an unweighted automaton, exactly the same strings in $(\Sigma \times K)^*$ are accepted from q and from r. So merging will merge q and r, and $\alpha \overset{\bar{M}}{\sim} \beta$ as claimed.

[25] It is not an equivalence relation on all of Σ^*, since $\alpha \notin D$ is related by $\overset{F}{\sim}$ to *every* β. This corresponds to the fact that a dead state can be made to merge with any state by pushing $\underline{0}$ back from it, so that the arcs to it have weight $\underline{0}$ and the arcs from it have arbitrary weights. Our construction of $\bar{M}$ only creates states for the equivalence classes of D; $\delta(0, \alpha)$ for $\alpha \notin D$ is undefined, not a dead state.

[26] We must check that we did not divide by $\underline{0}$ and obtain a false equation. It suffices to show that $k \neq \underline{0}$ and $\lambda(F_q) \neq \underline{0}$. Fortunately, $\alpha \in D$ implies both. (It implies $F_q \neq \underline{0}$, so $(\gamma^{-1}F_q)(\varepsilon) = F_q(\gamma) \neq \underline{0}$ for some γ. Hence $\lambda(F_q) \neq \underline{0}$ since otherwise $\lambda(\gamma^{-1}F_q) = \underline{0}$ and $\lambda(\gamma^{-1}F_q) \backslash (\gamma^{-1}F_q)(\varepsilon)$ is undefined, contradicting the final-quotient property.)