CHAPTER 4

# *Statistical Natural Language Processing*



## 4.0. Introduction

The application of statistical methods to natural language processing has been remarkably successful over the past two decades. The wide availability of text and speech corpora has played a critical role in their success since, as for all learning techniques, these methods heavily rely on data. Many of the components of complex natural language processing systems, e.g., text normalizers, morphological or phonological analyzers, part-of-speech taggers, grammars or language models, pronunciation models, context-dependency models, acoustic Hidden-Markov Models (HMMs), are statistical models derived from large data sets using modern learning techniques. These models are often given as *weighted automata* or *weighted finite-state transducers* either directly or as a result of the approximation of more complex models.

Weighted automata and transducers are the finite automata and finite-state

| SEMIRING | SET | $\oplus$ | $\otimes$ | $\overline{0}$ | $\overline{1}$ |
|---|---|---|---|---|---|
| Boolean | $\{0,1\}$ | $\vee$ | $\wedge$ | 0 | 1 |
| Probability | $\mathbb{R}_+$ | $+$ | $\times$ | 0 | 1 |
| Log | $\mathbb{R} \cup \{-\infty, +\infty\}$ | $\oplus_{\log}$ | $+$ | $+\infty$ | 0 |
| Tropical | $\mathbb{R} \cup \{-\infty, +\infty\}$ | min | $+$ | $+\infty$ | 0 |

**Table 4.1.** *Semiring examples.* $\oplus_{\log}$ *is defined by:* $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$.

transducers described in Chapter 1 Section 1.5 with the addition of some weight to each transition. Thus, weighted finite-state transducers are automata in which each transition, in addition to its usual input label, is augmented with an *output label* from a possibly different alphabet, and carries some *weight*. The weights may correspond to probabilities or log-likelihoods or they may be some other costs used to rank alternatives. More generally, as we shall see in the next section, they are elements of a *semiring* set. Transducers can be used to define a mapping between two different types of information sources, e.g., word and phoneme sequences. The weights are crucial to model the uncertainty of such mappings. Weighted transducers can be used for example to assign different pronunciations to the same word but with different ranks or probabilities.

Novel algorithms are needed to combine and optimize large statistical models represented as weighted automata or transducers. This chapter reviews several recent weighted transducer algorithms, including composition of weighted transducers, determinization of weighted automata and minimization of weighted automata, which play a crucial role in the construction of modern statistical natural language processing systems. It also outlines their use in the design of modern real-time speech recognition systems. It discusses and illustrates the representation by weighted automata and transducers of the components of these systems, and describes the use of these algorithms for combining, searching, and optimizing large component transducers of several million transitions for creating real-time speech recognition systems.

## 4.1. Preliminaries

This section introduces the definitions and notation used in the following.

A system $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a *semiring* if $(\mathbb{K}, \oplus, \overline{0})$ is a commutative monoid with identity element $\overline{0}$, $(\mathbb{K}, \otimes, \overline{1})$ is a monoid with identity element $\overline{1}$, $\otimes$ distributes over $\oplus$, and $\overline{0}$ is an annihilator for $\otimes$: for all $a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$. Thus, a semiring is a ring that may lack negation. Table 4.1 lists some familiar semirings. In addition to the Boolean semiring, and the probability semiring used to combine probabilities, two semirings often used in text and speech processing applications are the *log semiring* which is isomorphic to the probability semiring via the negative-log morphism, and the *tropical semiring* which is derived from the log semiring using the *Viterbi approximation*. A *left semiring* is a system that verifies all the axioms of a semiring except from the right ditributivity. In the following definitions, $\mathbb{K}$ will be used to denote a left semiring or a

semiring.

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

A *weighted finite-state transducer* $\mathfrak{T}$ over a semiring $\mathbb{K}$ is an 8-tuple $\mathfrak{T} = (\mathcal{A}, \mathcal{B}, Q, I, F, E, \lambda, \rho)$ where: $\mathcal{A}$ is the finite input alphabet of the transducer; $\mathcal{B}$ is the finite output alphabet; $Q$ is a finite set of states; $I \subseteq Q$ the set of initial states; $F \subseteq Q$ the set of final states; $E \subseteq Q \times (\mathcal{A} \cup \{\varepsilon\}) \times (\mathcal{B} \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ a finite set of transitions; $\lambda : I \to \mathbb{K}$ the initial weight function; and $\rho : F \to \mathbb{K}$ the final weight function mapping $F$ to $\mathbb{K}$. $E[q]$ denotes the set of transitions leaving a state $q \in Q$. $|\mathfrak{T}|$ denotes the sum of the number of states and transitions of $\mathfrak{T}$.

*Weighted automata* are defined in a similar way by simply omitting the input or output labels. Let $\Pi_1(\mathfrak{T})$ ($\Pi_2(\mathfrak{T})$) denote the weighted automaton obtained from a weighted transducer $\mathfrak{T}$ by omitting the input (resp. output) labels of $\mathfrak{T}$.

Given a transition $e \in E$, let $p[e]$ denote its origin or previous state, $n[e]$ its destination state or next state, $i[e]$ its input label, $o[e]$ its output label, and $w[e]$ its weight. A *path* $\pi = e_1 \cdots e_k$ is an element of $E^*$ with consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. $n$, $p$, and $w$ can be extended to paths by setting: $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$ and by defining the weight of a path as the $\otimes$-product of the weights of its constituent transitions: $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. More generally, $w$ is extended to any finite set of paths $R$ by setting: $w[R] = \bigoplus_{\pi \in R} w[\pi]$. Let $P(q, q')$ denote the set of paths from $q$ to $q'$ and $P(q, x, y, q')$ the set of paths from $q$ to $q'$ with input label $x \in \mathcal{A}^*$ and output label $y \in \mathcal{B}^*$. These definitions can be extended to subsets $R, R' \subseteq Q$, by: $P(R, x, y, R') = \cup_{q \in R,\, q' \in R'} P(q, x, y, q')$. A transducer $\mathfrak{T}$ is *regulated* if the weight associated by $\mathfrak{T}$ to any pair of input-output string $(x, y)$ given by:

$$[\![\mathfrak{T}]\!](x, y) = \bigoplus_{\pi \in P(I, x, y, F)} \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]] \tag{4.1.1}$$

is well-defined and in $\mathbb{K}$. $[\![\mathfrak{T}]\!](x, y) = \overline{0}$ when $P(I, x, y, F) = \emptyset$. In particular, when it does not have any $\varepsilon$-cycle, $\mathfrak{T}$ is always regulated.

## 4.2. Algorithms

### 4.2.1. Composition

Composition is a fundamental algorithm used to create complex weighted transducers from simpler ones. It is a generalization of the composition algorithm

presented in Chapter 1 Section 1.5 for unweighted finite-state transducers. Let $\mathbb{K}$ be a commutative semiring and let $\mathfrak{T}_1$ and $\mathfrak{T}_2$ be two weighted transducers defined over $\mathbb{K}$ such that the input alphabet of $\mathfrak{T}_2$ coincides with the output alphabet of $\mathfrak{T}_1$. Assume that the infinite sum $\bigoplus_z \mathfrak{T}_1(x,z) \otimes \mathfrak{T}_2(z,y)$ is well-defined and in $\mathbb{K}$ for all $(x,y) \in \mathcal{A}^* \times \mathcal{C}^*$. This condition holds for all transducers defined over a *closed semiring* such as the Boolean semiring and the tropical semiring and for all acyclic transducers defined over an arbitrary semiring. Then, the result of the composition of $\mathfrak{T}_1$ and $\mathfrak{T}_2$ is a weighted transducer denoted by $\mathfrak{T}_1 \circ \mathfrak{T}_2$ and defined for all $x, y$ by:

$$[\![\mathfrak{T}_1 \circ \mathfrak{T}_2]\!](x,y) = \bigoplus_z \mathfrak{T}_1(x,z) \otimes \mathfrak{T}_2(z,y) \tag{4.2.1}$$

Note that we use a *matrix notation* for the definition of composition as opposed to a *functional notation*. There exists a general and efficient composition algorithm for weighted transducers. States in the composition $\mathfrak{T}_1 \circ \mathfrak{T}_2$ of two weighted transducers $\mathfrak{T}_1$ and $\mathfrak{T}_2$ are identified with pairs of a state of $\mathfrak{T}_1$ and a state of $\mathfrak{T}_2$. Leaving aside transitions with $\varepsilon$ inputs or outputs, the following rule specifies how to compute a transition of $\mathfrak{T}_1 \circ \mathfrak{T}_2$ from appropriate transitions of $\mathfrak{T}_1$ and $\mathfrak{T}_2$:

$$(q_1, a, b, w_1, q_2) \text{ and } (q_1', b, c, w_2, q_2') \Longrightarrow ((q_1, q_1'), a, c, w_1 \otimes w_2, (q_2, q_2')) \tag{4.2.2}$$

The following is the pseudocode of the algorithm in the $\varepsilon$-free case.

```
WEIGHTED-COMPOSITION(T1, T2)
 1  Q ← I1 × I2
 2  S ← I1 × I2
 3  while S ≠ ∅ do
 4       (q1, q2) ← HEAD(S)
 5       DEQUEUE(S)
 6       if (q1, q2) ∈ I1 × I2 then
 7            I ← I ∪ {(q1, q2)}
 8            λ(q1, q2) ← λ1(q1) ⊗ λ2(q2)
 9       if (q1, q2) ∈ F1 × F2 then
10            F ← F ∪ {(q1, q2)}
11            ρ(q1, q2) ← ρ1(q1) ⊗ ρ2(q2)
12       for each (e1, e2) ∈ E[q1] × E[q2] such that o[e1] = i[e2] do
13            if (n[e1], n[e2]) ∉ Q then
14                 Q ← Q ∪ {(n[e1], n[e2])}
15                 ENQUEUE(S, (n[e1], n[e2]))
16            E ← E ∪ {((q1, q2), i[e1], o[e2], w[e1] ⊗ w[e2], (n[e1], n[e2]))}
17  return T
```

The algorithm takes as input $\mathfrak{T}_1 = (\mathcal{A}, \mathcal{B}, Q_1, I_1, F_1, E_1, \lambda_1, \rho_1)$ and $\mathfrak{T}_2 = (\mathcal{B}, \mathcal{C}, Q_2, I_2, F_2, E_2, \lambda_2, \rho_2)$, two weighted transducers, and outputs a weighted

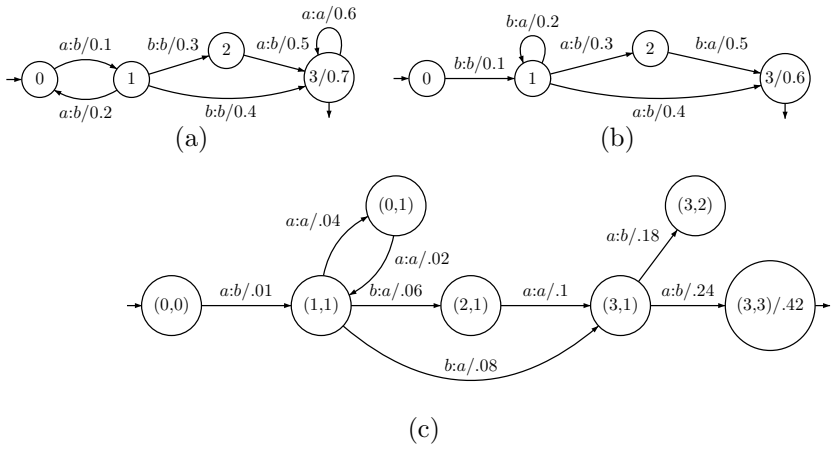


**Figure 4.1.** (a) Weighted transducer $\mathfrak{T}_1$ over the probabilityl semiring. (b) Weighted transducer $\mathfrak{T}_2$ over the probability semiring. (c) Composition of $\mathfrak{T}_1$ and $\mathfrak{T}_2$. Initial states are represented by an incoming arrow, final states with an outgoing arrow. Inside each circle, the first number indicates the state number, the second, at final states only, the value of the final weight function $\rho$ at that state. Arrows represent transitions and are labeled with symbols followed by their corresponding weight.

transducer $\mathfrak{T} = (\mathcal{A}, \mathcal{C}, Q, I, F, E, \lambda, \rho)$ realizing the composition of $\mathfrak{T}_1$ and $\mathfrak{T}_2$. $E$, $I$, and $F$ are all assumed to be initialized to the empty set.

The algorithm uses a queue $S$ containing the set of pairs of states yet to be examined. The queue discipline of $S$ can be arbitrarily chosen and does not affect the termination of the algorithm. The set of states $Q$ is originally reduced to the set of pairs of the initial states of the original transducers and $S$ is initialized to the same (lines 1-2). Each time through the loop of lines 3-16, a new pair of states $(q_1, q_2)$ is extracted from $S$ (lines 4-5). The initial weight of $(q_1, q_2)$ is computed by $\otimes$-multiplying the initial weights of $q_1$ and $q_2$ when they are both initial states (lines 6-8). Similar steps are followed for final states (lines 9-11). Then, for each pair of matching transitions $(e_1, e_2)$, a new transition is created according to the rules specified earlier (line 16). If the destination state $(n[e_1], n[e_2])$ has not been found before, it is added to $Q$ and inserted in $S$ (lines 14-15).

In the worst case, all transitions of $\mathfrak{T}_1$ leaving a state $q_1$ match all those of $\mathfrak{T}_2$ leaving state $q_1'$, thus the space and time complexity of composition is quadratic: $O(|\mathfrak{T}_1||\mathfrak{T}_2|)$. However, a lazy implementation of composition can be used to construct just the part of the composed transducer that is needed. Figures 4.1(a)-(c) illustrate the algorithm when applied to the transducers of Figures 4.1(a)-(b) defined over the probability semiring.

More care is needed when $\mathfrak{T}_1$ admits output $\varepsilon$ labels and $\mathfrak{T}_2$ input $\varepsilon$ labels. Indeed, as illustrated by Figure 4.2, a straightforward generalization of the $\varepsilon$-

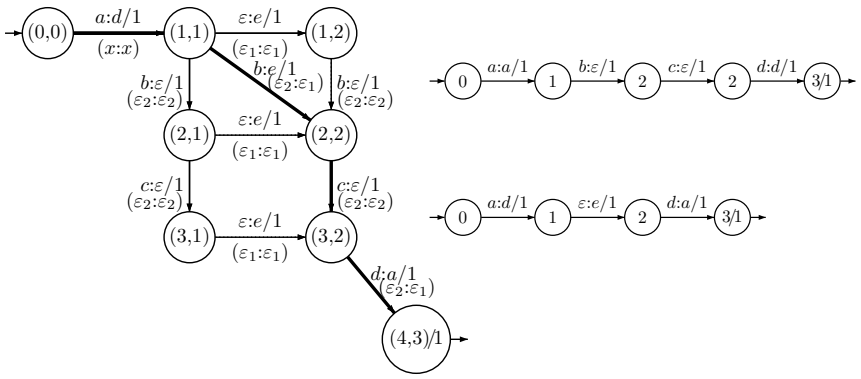


**Figure 4.2.** Redundant $\varepsilon$-paths. A straightforward generalization of the $\varepsilon$-free case could generate all the paths from $(1,1)$ to $(3,2)$ when composing the two simple transducers on the right-hand side.

free case would generate redundant $\varepsilon$-paths and, in the case of non-idempotent semirings, would lead to an incorrect result. The weight of the matching paths of the original transducers would be counted $p$ times, where $p$ is the number of redundant paths in the result of composition.

To cope with this problem, all but one $\varepsilon$-path must be filtered out of the composite transducer. Figure 4.2 indicates in boldface one possible choice for that path, which in this case is the shortest. Remarkably, that filtering mechanism can be encoded as a finite-state transducer.

Let $\tilde{\mathfrak{T}}_1$ ($\tilde{\mathfrak{T}}_2$) be the weighted transducer obtained from $\mathfrak{T}_1$ (resp. $\mathfrak{T}_2$) by replacing the output (resp. input) $\varepsilon$ labels with $\varepsilon_2$ (resp. $\varepsilon_1$), and let $\mathfrak{F}$ be the filter finite-state transducer represented in Figure 4.3. Then $\tilde{\mathfrak{T}}_1 \circ \mathfrak{F} \circ \tilde{\mathfrak{T}}_2 = \mathfrak{T}_1 \circ \mathfrak{T}_2$. Since the two compositions in $\tilde{\mathfrak{T}}_1 \circ \mathfrak{F} \circ \tilde{\mathfrak{T}}_2$ do not involve $\varepsilon$'s, the $\varepsilon$-free composition already described can be used to compute the resulting transducer.

Intersection (or Hadamard product) of weighted automata and composition of finite-state transducers are both special cases of composition of weighted transducers. Intersection corresponds to the case where input and output labels of transitions are identical and composition of unweighted transducers is obtained simply by omitting the weights.

In general, the definition of composition cannot be extended to the case of non-commutative semirings because the composite transduction cannot always be represented by a weighted finite-state transducer. Consider for example, the case of two transducers $\mathfrak{T}_1$ and $\mathfrak{T}_2$ accepting the same set of strings $(a,a)^*$, with $[\![\mathfrak{T}_1]\!](a,a) = x \in \mathbb{K}$ and $[\![\mathfrak{T}_2]\!](a,a) = y \in \mathbb{K}$ and let $\tau$ be the composite of the transductions corresponding to $\mathfrak{T}_1$ and $\mathfrak{T}_2$. Then, for any non-negative integer $n$, $\tau(a^n, a^n) = x^n \otimes y^n$ which in general is different from $(x \otimes y)^n$ if $x$ and $y$

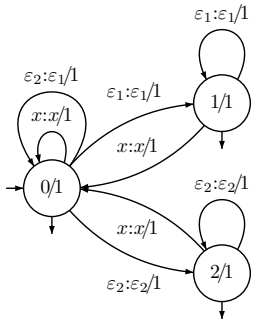


**Figure 4.3.** Filter for composition $\mathfrak{F}$.

do not commute. An argument similar to the classical Pumping lemma can then be used to show that $\tau$ cannot be represented by a weighted finite-state transducer.

When $\mathfrak{T}_1$ and $\mathfrak{T}_2$ are acyclic, composition can be extended to the case of non-commutative semirings. The algorithm would then consist of matching paths of $\mathfrak{T}_1$ and $\mathfrak{T}_2$ directly rather than matching their constituent transitions. The termination of the algorithm is guaranteed by the fact that the number of paths of $\mathfrak{T}_1$ and $\mathfrak{T}_2$ is finite. However, the time and space complexity of the algorithm is then exponential.

The weights of matching transitions and paths are $\otimes$-multiplied in composition. One might wonder if another useful operation, $\times$, can be used instead of $\otimes$, in particular when $\mathbb{K}$ is not commutative. The following proposition proves that that cannot be.

PROPOSITION 4.2.1. *Let $(\mathbb{K}, \times, e)$ be a monoid. Assume that $\times$ is used instead of $\otimes$ in composition. Then, $\times$ coincides with $\otimes$ and $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a commutative semiring.*

*Proof.* Consider two sets of consecutive transitions of two paths: $\pi_1 = (p_1, a, a, x, q_1)(q_1, b, b, y, r_1)$ and $\pi_2 = (p_2, a, a, u, q_2)(q_2, b, b, v, r_2)$. Matching these transitions using $\times$ result in the following:

$$((p_1, p_2), a, a, x \times u, (q_1, q_2)) \quad \text{and} \quad ((q_1, q_2), b, b, y \times v, (r_1, r_2)) \tag{4.2.3}$$

Since the weight of the path obtained by matching $\pi_1$ and $\pi_2$ must also correspond to the $\times$-multiplication of the weight of $\pi_1$, $x \otimes y$, and the weight of $\pi_2$, $u \otimes v$, we have:

$$(x \times u) \otimes (y \times v) = (x \otimes y) \times (u \otimes v) \tag{4.2.4}$$

This identity must hold for all $x, y, u, v \in \mathbb{K}$. Setting $u = y = e$ and $v = \overline{1}$ leads to $x = x \otimes e$ and similarly $x = e \otimes x$ for all $x$. Since the identity element of $\otimes$ is unique, this proves that $e = \overline{1}$.

With $u = y = \overline{1}$, identity 4.2.4 can be rewritten as: $x \otimes v = x \times v$ for all $x$ and $v$, which shows that $\times$ coincides with $\otimes$. Finally, setting $x = v = \overline{1}$ gives $u \otimes y = y \times u$ for all $y$ and $u$ which shows that $\otimes$ is commutative. ∎

### 4.2.2. Determinization

This section describes a generic determinization algorithm for weighted automata. It is thus a generalization of the determinization algorithm for unweighted finite automata. When combined with the (unweighted) determinization for finite-state transducers presented in Chapter 1 Section 1.5, it leads to an algorithm for determinizing weighted transducers.[1]

A weighted automaton is said to be *deterministic* or *subsequential* if it has a unique initial state and if no two transitions leaving any state share the same input label. There exists a natural extension of the classical subset construction to the case of weighted automata over a weakly left divisible left semiring called *determinization*.[2] The algorithm is generic: it works with any weakly left divisible semiring. The pseudocode of the algorithm is given below with $Q'$, $I'$, $F'$, and $E'$ all initialized to the empty set.

WEIGHTED-DETERMINIZATION($\mathfrak{A}$)

$i' \leftarrow \{(i, \lambda(i)) : i \in I\}$
$\lambda'(i') \leftarrow \overline{1}$
$S \leftarrow \{i'\}$
**while** $S \neq \emptyset$ **do**
  $p' \leftarrow$ HEAD($S$)
  DEQUEUE($S$)
  **for** each $x \in i[E[Q[p']]]$ **do**
    $w' \leftarrow \bigoplus \{v \otimes w : (p, v) \in p', (p, x, w, q) \in E\}$
    $q' \leftarrow \{(q, \bigoplus \{w'^{-1} \otimes (v \otimes w) : (p, v) \in p', (p, x, w, q) \in E\}) :$
      $q = n[e], i[e] = x, e \in E[Q[p']]\}$
    $E' \leftarrow E' \cup \{(p', x, w', q')\}$
    **if** $q' \notin Q'$ **then**
      $Q' \leftarrow Q' \cup \{q'\}$
      **if** $Q[q'] \cap F \neq \emptyset$ **then**
        $F' \leftarrow F' \cup \{q'\}$
        $\rho'(q') \leftarrow \bigoplus \{v \otimes \rho(q) : (q, v) \in q', q \in F\}$
      ENQUEUE($S, q'$)
**return** $\mathfrak{T}'$

[1] In reality, the determinization of unweighted and that of weighted finite-state transducers can both be viewed as special instances of the generic algorithm presented here but, for clarity purposes, we will not emphasize that view in what follows.

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

A *weighted subset* $p'$ of $Q$ is a set of pairs $(q, x) \in Q \times \mathbb{K}$. $Q[p']$ denotes the set of states $q$ of the weighted subset $p'$. $E[Q[p']]$ represents the set of transitions leaving these states, and $i[E[Q[p']]]$ the set of input labels of these transitions.

The states of the output automaton can be identified with (weighted) subsets of the states of the original automaton. A state $r$ of the output automaton that can be reached from the start state by a path $\pi$ is identified with the set of pairs $(q, x) \in Q \times \mathbb{K}$ such that $q$ can be reached from an initial state of the original machine by a path $\sigma$ with $i[\sigma] = i[\pi]$ and $\lambda[p[\sigma]] \otimes w[\sigma] = \lambda[p[\pi]] \otimes w[\pi] \otimes x$. Thus, $x$ can be viewed as the *residual* weight at state $q$. When it terminates, the algorithm takes as input a weighted automaton $\mathfrak{A} = (\mathcal{A}, Q, I, F, E, \lambda, \rho)$ and yields an equivalent subsequential weighted automaton $\mathfrak{A}' = (\mathcal{A}, Q', I', F', E', \lambda', \rho')$.

The algorithm uses a queue $S$ containing the set of states of the resulting automaton $\mathfrak{A}'$, yet to be examined. The queue discipline of $S$ can be arbitrarily chosen and does not affect the termination of the algorithm. $\mathfrak{A}'$ admits a unique initial state, $i'$, defined as the set of initial states of $\mathfrak{A}$ augmented with their respective initial weights. Its input weight is $\overline{1}$ (lines 1-2). $S$ originally contains only the subset $i'$ (line 3). Each time through the loop of lines 4-16, a new subset $p'$ is extracted from $S$ (lines 5-6). For each $x$ labeling at least one of the transitions leaving a state $p$ of the subset $p'$, a new transition with input label $x$ is constructed. The weight $w'$ associated to that transition is the sum of the weights of all transitions in $E[Q[p']]$ labeled with $x$ pre-$\otimes$-multiplied by the residual weight $v$ at each state $p$ (line 8). The destination state of the transition is the subset containing all the states $q$ reached by transitions in $E[Q[p']]$ labeled with $x$. The weight of each state $q$ of the subset is obtained by taking the $\oplus$-sum of the residual weights of the states $p$ $\otimes$-times the weight of the transition from $p$ leading to $q$ and by *dividing* that by $w'$. The new subset $q'$ is inserted in the queue $S$ when it is a new state (line 15). If any of the states in the subset $q'$ is final, $q'$ is made a final state and its final weight is obtained by summing the final weights of all the final states in $q'$, pre-$\otimes$-multiplied by their residual weight $v$ (line 14).

Figures 4.4(a)-(b) illustrate the determinization of a weighted automaton over the tropical semiring. The worst case complexity of determinization is exponential even in the unweighted case. However, in many practical cases such as for weighted automata used in large-vocabulary speech recognition, this blow-up does not occur. It is also important to notice that just like composition, determinization admits a natural lazy implementation which can be useful for saving space.

Unlike the unweighted case, determinization does not halt on all input weighted automata. In fact, some weighted automata, non *subsequentiable* automata, do not even admit equivalent subsequential machines. But even for some subsequentiable automata, the algorithm does not halt. We say that a weighted automaton $\mathfrak{A}$ is *determinizable* if the determinization algorithm halts for the input $\mathfrak{A}$. With a determinizable input, the algorithm outputs an equivalent subsequential weighted automaton.

There exists a general *twins property* for weighted automata that provides a

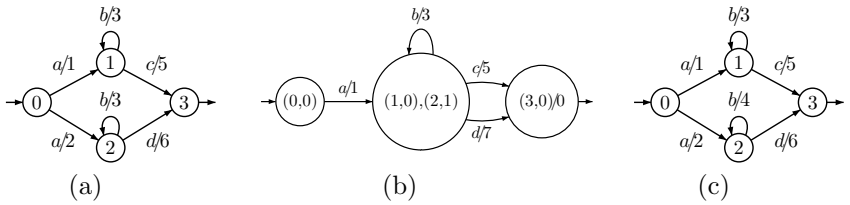


**Figure 4.4.** Determinization of weighted automata. (a) Weighted automaton over the tropical semiring $\mathfrak{A}$. (b) Equivalent weighted automaton $\mathfrak{B}$ obtained by determinization of $\mathfrak{A}$. (c) Non-determinizable weighted automaton over the tropical semiring, states 1 and 2 are non-twin siblings.

characterization of determinizable weighted automata under some general conditions. Let $\mathfrak{A}$ be a weighted automaton over a weakly left divisible left semiring $\mathbb{K}$. Two states $q$ and $q'$ of $\mathfrak{A}$ are said to be *siblings* if there exist two strings $x$ and $y$ in $\mathcal{A}^*$ such that both $q$ and $q'$ can be reached from $I$ by paths labeled with $x$ and there is a cycle at $q$ and a cycle at $q'$ both labeled with $y$. When $\mathbb{K}$ is a commutative and cancellative semiring, two sibling states are said to be *twins* iff for any string $y$:

$$w[P(q, y, q)] = w[P(q', y, q')] \tag{4.2.5}$$

$\mathfrak{A}$ has *the twins property* if any two sibling states of $\mathfrak{A}$ are twins. Figure 4.4(c) shows an unambiguous weighted automaton over the tropical semiring that does not have the twins property: states 1 and 2 can be reached by paths labeled with $a$ from the initial state and admit cycles with the same label $b$, but the weights of these cycles (3 and 4) are different.

THEOREM 4.2.2. *Let $\mathfrak{A}$ be a weighted automaton over the tropical semiring. If $\mathfrak{A}$ has the twins property, then $\mathfrak{A}$ is determinizable.*

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

THEOREM 4.2.3. *Let $\mathfrak{A}$ be a trim unambiguous weighted automaton over the tropical semiring. Then the three following properties are equivalent:*

1. *$\mathfrak{A}$ is determinizable.*
2. *$\mathfrak{A}$ has the twins property.*
3. *$\mathfrak{A}$ is subsequentiable.*

There exists an efficient algorithm for testing the twins property for weighted automata, which cannot be presented briefly in this chapter. Note that any acyclic weighted automaton over a zero-sum-free semiring has the twins property and is determinizable.

### 4.2.3. Weight pushing

The choice of the distribution of the total weight along each successful path of a weighted automaton does not affect the definition of the function realized by that automaton, but this may have a critical impact on the efficiency in many applications, e.g., natural language processing applications, when a heuristic pruning is used to visit only a subpart of the automaton. There exists an algorithm, *weight pushing*, for normalizing the distribution of the weights along the paths of a weighted automaton or more generally a weighted directed graph. The transducer normalization algorithm presented in Chapter 1 Section 1.5 can be viewed as a special instance of this algorithm.

Let $\mathfrak{A}$ be a weighted automaton over a semiring $\mathbb{K}$. Assume that $\mathbb{K}$ is zero-sum-free and weakly left divisible. For any state $q \in Q$, assume that the following sum is well-defined and in $\mathbb{K}$:

$$d[q] = \bigoplus_{\pi \in P(q,F)} (w[\pi] \otimes \rho[n[\pi]]) \tag{4.2.6}$$

$d[q]$ is the *shortest-distance* from $q$ to $F$. $d[q]$ is well-defined for all $q \in Q$ when $\mathbb{K}$ is a $k$-closed semiring. The weight pushing algorithm consists of computing each shortest-distance $d[q]$ and of *reweighting* the transition weights, initial weights and final weights in the following way:

$$\forall e \in E \text{ s.t. } d[p[e]] \neq \overline{0}, w[e] \leftarrow d[p[e]]^{-1} \otimes w[e] \otimes d[n[e]] \tag{4.2.7}$$

$$\forall q \in I, \lambda[q] \leftarrow \lambda[q] \otimes d[q] \tag{4.2.8}$$

$$\forall q \in F, \text{ s.t. } d[q] \neq \overline{0}, \rho[q] \leftarrow d[q]^{-1} \otimes \rho[q] \tag{4.2.9}$$

Each of these operations can be assumed to be done in constant time, thus reweighting can be done in linear time $O(\mathfrak{T}_{\otimes}|\mathfrak{A}|)$ where $\mathfrak{T}_{\otimes}$ denotes the worst cost of an $\otimes$-operation. The complexity of the computation of the shortest-distances depends on the semiring. In the case of $k$-closed semirings such as the tropical semiring, $d[q]$, $q \in Q$, can be computed using a generic shortest-distance algorithm. The complexity of the algorithm is linear in the case of an acyclic automaton: $O(\text{Card}(Q)+(\mathfrak{T}_{\oplus}+\mathfrak{T}_{\otimes})\,\text{Card}(E))$, where $\mathfrak{T}_{\oplus}$ denotes the worst cost of an $\oplus$-operation. In the case of a general weighted automaton over the tropical semiring, the complexity of the algorithm is $O(\text{Card}(E)+\text{Card}(Q)\log\text{Card}(Q))$.

In the case of closed semirings such as $(\mathbb{R}_+, +, \times, 0, 1)$, a generalization of the Floyd-Warshall algorithm for computing all-pairs shortest-distances can be used. The complexity of the algorithm is $\Theta(\text{Card}(Q)^3(\mathfrak{T}_{\oplus}+\mathfrak{T}_{\otimes}+\mathfrak{T}_{*}))$ where $\mathfrak{T}_{*}$ denotes the worst cost of the closure operation. The space complexity of these algorithms is $\Theta(\text{Card}(Q)^2)$. These complexities make it impractical to use the Floyd-Warshall algorithm for computing $d[q]$, $q \in Q$, for relatively large graphs or automata of several hundred million states or transitions. An approximate version of a generic shortest-distance algorithm can be used instead to compute $d[q]$ efficiently.

Roughly speaking, the algorithm *pushes the weights* of each path as much as possible towards the initial states. Figures 4.5(a)-(c) illustrate the application of the algorithm in a special case both for the tropical and probability semirings.

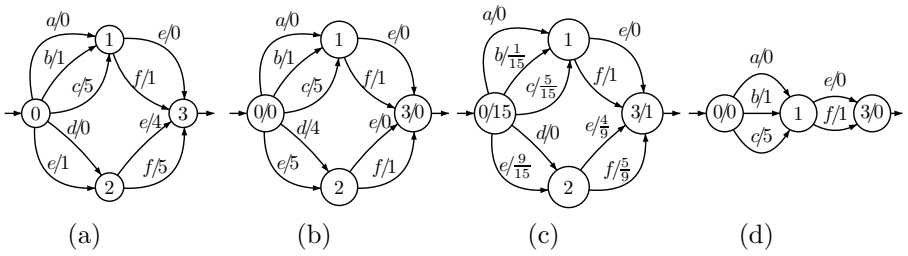


**Figure 4.5.** Weight pushing algorithm. (a) Weighted automaton $\mathfrak{A}$. (b) Equivalent weighted automaton $\mathfrak{B}$ obtained by weight pushing in the tropical semiring. (c) Weighted automaton $\mathfrak{C}$ obtained from $\mathfrak{A}$ by weight pushing in the probability semiring. (d) Minimal weighted automaton over the tropical semiring equivalent to $\mathfrak{A}$.

Note that if $d[q] = \overline{0}$, then, since $\mathbb{K}$ is zero-sum-free, the weight of all paths from $q$ to $F$ is $\overline{0}$. Let $\mathfrak{A}$ be a weighted automaton over the semiring $\mathbb{K}$. Assume that $\mathbb{K}$ is closed or $k$-closed and that the shortest-distances $d[q]$ are all well-defined and in $\mathbb{K} - \{\overline{0}\}$. Note that in both cases we can use the distributivity over the infinite sums defining shortest distances. Let $e'$ ($\pi'$) denote the transition $e$ (path $\pi$) after application of the weight pushing algorithm. $e'$ ($\pi'$) differs from $e$ (resp. $\pi$) only by its weight. Let $\lambda'$ denote the new initial weight function, and $\rho'$ the new final weight function.

PROPOSITION 4.2.4. *Let $\mathfrak{B} = (\mathcal{A}, Q, I, F, E', \lambda', \rho')$ be the result of the weight pushing algorithm applied to the weighted automaton $\mathfrak{A}$, then*

1. *the weight of a successful path $\pi$ is unchanged after application of weight pushing:*

$$\lambda'[p[\pi']] \otimes w[\pi'] \otimes \rho'[n[\pi']] = \lambda[p[\pi]] \otimes w[\pi] \otimes \rho[n[\pi]] \tag{4.2.10}$$

2. *the weighted automaton $\mathfrak{B}$ is* stochastic, *i.e.*

$$\forall q \in Q, \bigoplus_{e' \in E'[q]} w[e'] = \overline{1} \tag{4.2.11}$$

*Proof.* Let $\pi' = e'_1 \dots e'_k$. By definition of $\lambda'$ and $\rho'$,

$$\begin{aligned}\lambda'[p[\pi']] \otimes w[\pi'] \otimes \rho'[n[\pi']] &= \lambda[p[e_1]] \otimes d[p[e_1]] \otimes d[p[e_1]]^{-1} \otimes w[e_1] \otimes d[n[e_1]] \otimes \cdots \\ &\quad \otimes d[p[e_k]]^{-1} \otimes w[e_k] \otimes d[n[e_k]] \otimes d[n[e_k]]^{-1} \otimes \rho[n[\pi]] \\ &= \lambda[p[\pi]] \otimes w[e_1] \otimes \cdots \otimes w[e_k] \otimes \rho[n[\pi]]\end{aligned}$$

which proves the first statement of the proposition. Let $q \in Q$,

$$\begin{aligned}\bigoplus_{e' \in E'[q]} w[e'] &= \bigoplus_{e \in E[q]} d[q]^{-1} \otimes w[e] \otimes d[n[e]] \\ &= d[q]^{-1} \otimes \bigoplus_{e \in E[q]} w[e] \otimes d[n[e]]\end{aligned}$$

$$
\begin{aligned}
&= d[q]^{-1} \otimes \bigoplus_{e \in E[q]} w[e] \otimes \bigoplus_{\pi \in P(n[e], F)} (w[\pi] \otimes \rho[n[\pi]]) \\
&= d[q]^{-1} \otimes \bigoplus_{e \in E[q], \pi \in P(n[e], F)} (w[e] \otimes w[\pi] \otimes \rho[n[\pi]]) \\
&= d[q]^{-1} \otimes d[q] = \overline{1}
\end{aligned}
$$

where we used the distributivity of the multiplicative operation over infinite sums in closed or $k$-closed semirings. This proves the second statement of the proposition. ∎

These two properties of weight pushing are illustrated by Figures 4.5(a)-(c): the total weight of a successful path is unchanged after pushing; at each state of the weighted automaton of Figure 4.5(b), the minimum weight of the outgoing transitions is 0, and at at each state of the weighted automaton of Figure 4.5(c), the weights of outgoing transitions sum to 1. Weight pushing can also be used to test the equivalence of two weighted automata.

### 4.2.4. Minimization

A deterministic weighted automaton is said to be *minimal* if there exists no other deterministic weighted automaton with a smaller number of states and realizing the same function. Two states of a deterministic weighted automaton are said to be *equivalent* if exactly the same set of strings with the same weights label paths from these states to a final state, the final weights being included. Thus, two equivalent states of a deterministic weighted automaton can be merged without affecting the function realized by that automaton. A weighted automaton is minimal when it admits no two distinct equivalent states after any redistribution of the weights along its paths.

There exists a general algorithm for computing a minimal deterministic automaton equivalent to a given weighted automaton. It is thus a generalization of the minimization algorithms for unweighted finite automata. It can be combined with the minimization algorithm for unweighted finite-state transducers presented in Chapter 1 Section 1.5 to minimize weighted finite-state transducers.[3] It consists of first applying the weight pushing algorithm to normalize the distribution of the weights along the paths of the input automaton, and then of treating each pair (label, weight) as a single label and applying the classical (unweighted) automata minimization.

THEOREM 4.2.5. *Let $\mathfrak{A}$ be a deterministic weighted automaton over a semiring $\mathbb{K}$. Assume that the conditions of application of the weight pushing algorithm hold, then the execution of the following steps:*

1. *weight pushing,*
2. *(unweighted) automata minimization,*

[3] In reality, the minimization of both unweighted and weighted finite-state transducers can be viewed as special instances of the algorithm presented here, but, for clarity purposes, we will not emphasize that view in what follows.

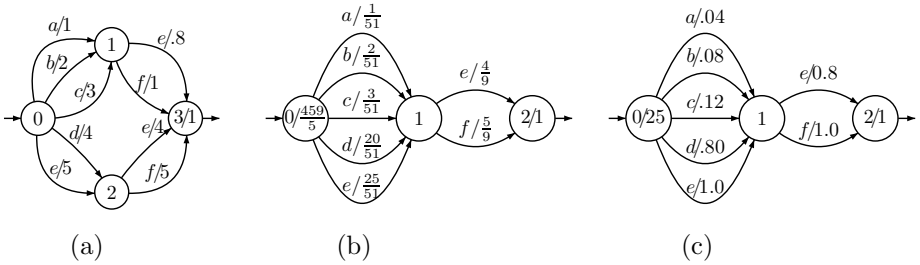


**Figure 4.6.** Minimization of weighted automata. (a) Weighted automaton $\mathfrak{A}'$ over the probability semiring. (b) Minimal weighted automaton $\mathfrak{B}'$ equivalent to $\mathfrak{A}'$. (c) Minimal weighted automaton $\mathfrak{C}'$ equivalent to $\mathfrak{A}'$.

*lead to a minimal weighted automaton equivalent to* $\mathfrak{A}$.

The complexity of automata minimization is linear in the case of acyclic automata $O(\text{Card}(Q) + \text{Card}(E))$ and in $O(\text{Card}(E) \log \text{Card}(Q))$ in the general case. Thus, in view of the complexity results given in the previous section, in the case of the tropical semiring, the total complexity of the weighted minimization algorithm is linear in the acyclic case $O(\text{Card}(Q) + \text{Card}(E))$ and in $O(\text{Card}(E) \log \text{Card}(Q))$ in the general case.

Figures 4.5(a), 4.5(b), and 4.5(d) illustrate the application of the algorithm in the tropical semiring. The automaton of Figure 4.5(a) cannot be further minimized using the classical unweighted automata minimization since no two states are equivalent in that machine. After weight pushing, the automaton (Figure 4.5(b)) has two states (1 and 2) that can be merged by the classical unweighted automata minimization.

Figures 4.6(a)-(c) illustrate the minimization of an automaton defined over the probability semiring. Unlike the unweighted case, a minimal weighted automaton is not unique, but all minimal weighted automata have the same graph topology, they only differ by the way the weights are distributed along each path. The weighted automata $\mathfrak{B}'$ and $\mathfrak{C}'$ are both minimal and equivalent to $\mathfrak{A}'$. $\mathfrak{B}'$ is obtained from $\mathfrak{A}'$ using the algorithm described above in the probability semiring and it is thus a stochastic weighted automaton in the probability semiring.

For a deterministic weighted automaton, the first operation of the semiring can be arbitrarily chosen without affecting the definition of the function it realizes. This is because, by definition, a deterministic weighted automaton admits at most one path labeled with any given string. Thus, in the algorithm described in theorem 4.2.5, the weight pushing step can be executed in any semiring $\mathbb{K}'$ whose multiplicative operation matches that of $\mathbb{K}$. The minimal weighted automata obtained by pushing the weights in $\mathbb{K}'$ is also minimal in $\mathbb{K}$ since it can be interpreted as a (deterministic) weighted automaton over $\mathbb{K}$.

In particular, $\mathfrak{A}'$ can be interpreted as a weighted automaton over the semiring $(\mathbb{R}_+, \max, \times, 0, 1)$. The application of the weighted minimization algorithm

to $\mathfrak{A}'$ in this semiring leads to the minimal weighted automaton $\mathfrak{C}'$ of Figure 4.6(c). $\mathfrak{C}'$ is also a *stochastic* weighted automaton in the sense that, at any state, the maximum weight of all outgoing transitions is one.

This fact leads to several interesting observations. One is related to the complexity of the algorithms. Indeed, we can choose a semiring $\mathbb{K}'$ in which the complexity of weight pushing is better than in $\mathbb{K}$. The resulting automaton is still minimal in $\mathbb{K}$ and has the additional property of being stochastic in $\mathbb{K}'$. It only differs from the weighted automaton obtained by pushing weights in $\mathbb{K}$ in the way weights are distributed along the paths. They can be obtained from each other by application of weight pushing in the appropriate semiring. In the particular case of a weighted automaton over the probability semiring, it may be preferable to use weight pushing in the $(\max, \times)$-semiring since the complexity of the algorithm is then equivalent to that of classical single-source shortest-paths algorithms. The corresponding algorithm is a special instance of the generic shortest-distance algorithm.

Another important point is that the weight pushing algorithm may not be defined in $\mathbb{K}$ because the machine is not zero-sum-free or for other reasons. But an alternative semiring $\mathbb{K}'$ can sometimes be used to minimize the input weighted automaton.

The results just presented were all related to the minimization of the number of states of a deterministic weighted automaton. The following proposition shows that minimizing the number of states coincides with minimizing the number of transitions.

PROPOSITION 4.2.6. *Let $\mathfrak{A}$ be a minimal deterministic weighted automaton, then $\mathfrak{A}$ has the minimal number of transitions.*

*Proof.* Let $\mathfrak{A}$ be a deterministic weighted automaton with the minimal number of transitions. If two distinct states of $\mathfrak{A}$ were equivalent, they could be merged, thereby strictly reducing the number of its transitions. Thus, $\mathfrak{A}$ must be a minimal deterministic automaton. Since, minimal deterministic automata have the same topology, in particular the same number of states and transitions, this proves the proposition.

## 4.3. Application to speech recognition

Much of the statistical techniques now widely used in natural language processing were inspired by early work in speech recognition. This section discusses the representation of the component models of an automatic speech recognition system by weighted transducers and describes how they can be combined, searched, and optimized using the algorithms described in the previous sections. The methods described can be used similarly in many other areas of natural language processing.

### 4.3.1. Statistical formulation

*Speech recognition* consists of generating accurate written transcriptions for spoken utterances. The desired transcription is typically a sequence of words, but it may also be the utterance's phonemic or syllabic transcription or a transcription into any other sequence of written units.

The problem can be formulated as a maximum-likelihood decoding problem, or the so-called *noisy channel* problem. Given a speech utterance, speech recognition consists of determining its most likely written transcription. Thus, if we let $o$ denote the observation sequence produced by a signal processing system, $w$ a (word) transcription sequence over an alphabet $\mathcal{A}$, and $\mathbf{P}(w \mid o)$ the probability of the transduction of $o$ into $w$, the problem consists of finding $\hat{w}$ as defined by:

$$\hat{w} = \underset{w \in \mathcal{A}^*}{\operatorname{argmax}} \mathbf{P}(w \mid o) \tag{4.3.1}$$

Using Bayes' rule, $\mathbf{P}(w \mid o)$ can be rewritten as: $\frac{\mathbf{P}(o|w)\mathbf{P}(w)}{\mathbf{P}(o)}$. Since $\mathbf{P}(o)$ does not depend on $w$, the problem can be reformulated as:

$$\hat{w} = \underset{w \in \mathcal{A}^*}{\operatorname{argmax}} \mathbf{P}(o \mid w)\, \mathbf{P}(w) \tag{4.3.2}$$

where $\mathbf{P}(w)$ is the a priori probability of the written sequence $w$ in the language considered and $\mathbf{P}(o \mid w)$ the probability of observing $o$ given that the sequence $w$ has been uttered. The probabilistic model used to estimate $\mathbf{P}(w)$ is called a *language model* or a *statistical grammar*. The generative model associated to $\mathbf{P}(o \mid w)$ is a combination of several knowledge sources, in particular the *acoustic model*, and the *pronunciation model*. $\mathbf{P}(o \mid w)$ can be decomposed into several intermediate levels e.g., that of phones, syllables, or other units. In most large-vocabulary speech recognition systems, it is decomposed into the following probabilistic models that are assumed independent:

- $\mathbf{P}(p \mid w)$, a pronunciation model or lexicon transducing word sequences $w$ to phonemic sequences $p$;
- $\mathbf{P}(c \mid p)$, a context-dependency transduction mapping phonemic sequences $p$ to context-dependent phone sequences $c$;
- $\mathbf{P}(d \mid c)$, a context-dependent phone model mapping sequences of context-dependent phones $c$ to sequences of distributions $d$; and
- $\mathbf{P}(o \mid d)$, an acoustic model applying distribution sequences $d$ to observation sequences.[4]

Since the models are assumed to be independent,

$$\mathbf{P}(o \mid w) = \sum_{d,c,p} \mathbf{P}(o \mid d)\mathbf{P}(d \mid c)\mathbf{P}(c \mid p)\mathbf{P}(p \mid w) \tag{4.3.3}$$

[4] $\mathbf{P}(o \mid d)\mathbf{P}(d \mid c)$ or $\mathbf{P}(o \mid d)\mathbf{P}(d \mid c)\mathbf{P}(c \mid p)$ is often called an *acoustic model*.

Equation 4.3.2 can thus be rewritten as:

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_{d,c,p} \mathbf{P}(o \mid d)\mathbf{P}(d \mid c)\mathbf{P}(c \mid p)\mathbf{P}(p \mid w)\mathbf{P}(w) \tag{4.3.4}$$

The following sections discuss the definition and representation of each of these models and that of the observation sequences in more detail. The transduction models are typically given either directly or as a result of an approximation as weighted finite-state transducers. Similarly, the language model is represented by a weighted automaton.

### 4.3.2. Statistical grammar

In some relatively restricted tasks, the language model for $\mathbf{P}(w)$ is based on an unweighted rule-based grammar. But, in most large-vocabulary tasks, the model is a weighted grammar derived from large corpora of several million words using statistical methods. The purpose of the model is to assign a probability to each sequence of words, thereby assigning a ranking to all sequences. Thus, the parsing information it may supply is not directly relevant to the statistical formulation described in the previous section.

The probabilistic model derived from corpora may be a probabilistic context-free grammmar. But, in general, context-free grammars are computationally too demanding for real-time speech recognition systems. The amount of work required to expand a recognition hypothesis can be unbounded for an unrestricted grammar. Instead, a regular approximation of a probabilistic context-free grammar is used. In most large-vocabulary speech recognition systems, the probabilistic model is in fact directly constructed as a weighted regular grammar and represents an $n$-gram model. Thus, this section concentrates on a brief description of these models.[5]

Regardless of the structure of the model, using the Bayes's rule, the probability of the word sequence $w = w_1 \cdots w_k$ can be written as the following product of conditional probabilities:

$$\mathbf{P}(w) = \prod_{i=1}^{k} \mathbf{P}(w_i \mid w_1 \cdots w_{i-1}) \tag{4.3.5}$$

An $n$-gram model is based on the Markovian assumption that the probability of the occurrence of a word only depends on the $n-1$ preceding words, that is, for $i = 1 \ldots n$:

$$\mathbf{P}(w_i \mid w_1 \cdots w_{i-1}) = \mathbf{P}(w_i \mid h_i) \tag{4.3.6}$$

where the conditioning history $h_i$ has length at most $n-1$: $|h_i| \leq n-1$. Thus,

$$\mathbf{P}(w) = \prod_{i=1}^{k} \mathbf{P}(w_i \mid h_i) \tag{4.3.7}$$

[5] Similar probabilistic models are designed for biological sequences (see Chapter 6).

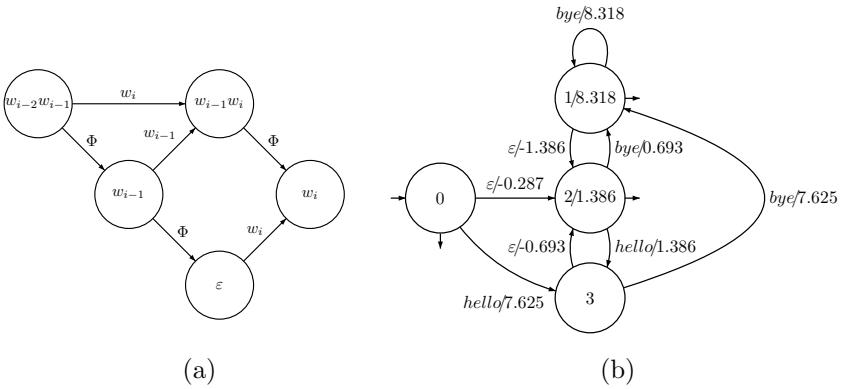


**Figure 4.7.** Katz back-off $n$-gram model. (a) Representation of a trigram model with failure transitions labeled with Φ. (b) Bigram model derived from the input text *hello bye bye*. The automaton is defined over the log semiring (the transition weights are negative log-probabilities). State 0 is the initial state. State 1 corresponds to the word *bye* and state 3 to the word *hello*. State 2 is the back-off state.

Let $c(w)$ denote the number of occurrences of a sequence $w$ in the corpus. $c(h_i)$ and $c(h_i w_i)$ can be used to estimate the conditional probability $\mathbf{P}(w_i \mid h_i)$. When $c(h_i) \neq 0$, the maximum likelihood estimate of $\mathbf{P}(w_i \mid h_i)$ is:

$$\hat{\mathbf{P}}(w_i \mid h_i) = \frac{c(h_i w_i)}{c(h_i)} \tag{4.3.8}$$

But, a classical data sparsity problem arises in the design of all $n$-gram models: the corpus, no matter how large, may contain no occurrence of $h_i$ ($c(h_i) = 0$). A solution to this problem is based on *smoothing techniques*. This consists of adjusting $\hat{\mathbf{P}}$ to reserve some probability mass for unseen $n$-gram sequences.

Let $\tilde{\mathbf{P}}(w_i \mid h_i)$ denote the adjusted conditional probability. A smoothing technique widely used in language modeling is the Katz back-off technique. The idea is to "back-off" to lower order $n$-gram sequences when $c(h_i w_i) = 0$. Define the backoff sequence of $h_i$ as the lower order $n$-gram sequence suffix of $h_i$ and denote it by $h_i'$. $h_i = u h_i'$ for some word $u$. Then, in a Katz back-off model, $\mathbf{P}(w_i \mid h_i)$ is defined as follows:

$$\mathbf{P}(w_i \mid h_i) = \begin{cases} \tilde{\mathbf{P}}(w_i \mid h_i) & \text{if } c(h_i w_i) > 0 \\ \alpha_{h_i} \mathbf{P}(w_i \mid h_i') & \text{otherwise} \end{cases} \tag{4.3.9}$$

where $\alpha_{h_i}$ is a factor ensuring normalization. The Katz back-off model admits a natural representation by a weighted automaton in which each state encodes a

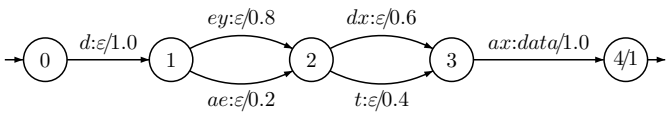


**Figure 4.8.** Section of a pronunciation model of English, a weighted transducer over the probability semiring giving a compact representation of four pronunciations of the word *data* due to two distinct pronunciations of the first vowel *a* and two pronunciations of the consonant *t* (flapped or not).

conditioning history of length less than $n$. As in the classical de Bruijn graphs, there is a transition labeled with $w_i$ from the state encoding $h_i$ to the state encoding $h'_i w_i$ when $c(h_i w_i) \neq 0$. A so-called *failure transition* can be used to capture the semantic of "otherwise" in the definition of the Katz back-off model and keep its representation compact. A failure transition is a transition taken at state $q$ when no other transition leaving $q$ has the desired label. Figure 4.3.2(a) illustrates that construction in the case of a trigram model ($n = 3$).

It is possible to give an explicit representation of these weighted automata without using failure transitions. However, the size of the resulting automata may become prohibitive. Instead, an approximation of that weighted automaton is used where failure transitions are simply replaced by $\varepsilon$-transitions. This turns out to cause only a very limited loss in accuracy.[6].

In practice, for numerical instability reasons negative-log probabilities are used and the language model weighted automaton is defined in the log semiring. Figure 4.3.2(b) shows the corresponding weighted automaton in a very simple case. We will denote by $\mathfrak{G}$ the weighted automaton representing the statistical grammar.

### 4.3.3. Pronunciation model

The representation of a pronunciation model $\mathbf{P}(p \mid w)$ (or lexicon) with weighted transducers is quite natural. Each word has a finite number of phonemic transcriptions. The probability of each pronunciation can be estimated from a corpus. Thus, for each word $x$, a simple weighted transducer $\mathfrak{T}_x$ mapping $x$ to its phonemic transcriptions can be constructed.

Figure 4.8 shows that representation in the case of the English word *data*. The closure of the union of the transducers $\mathfrak{T}_x$ for all the words $x$ considered gives a weighted transducer representation of the pronunciation model. We will denote by $\mathfrak{P}$ the equivalent transducer over the log semiring.

[6] An alternative when no offline optimization is used is to compute the explicit representation on-the-fly, as needed for the recognition of an utterance. There exists also a complex method for constructing an exact representation of an $n$-gram model which cannot be presented in this short chapter.

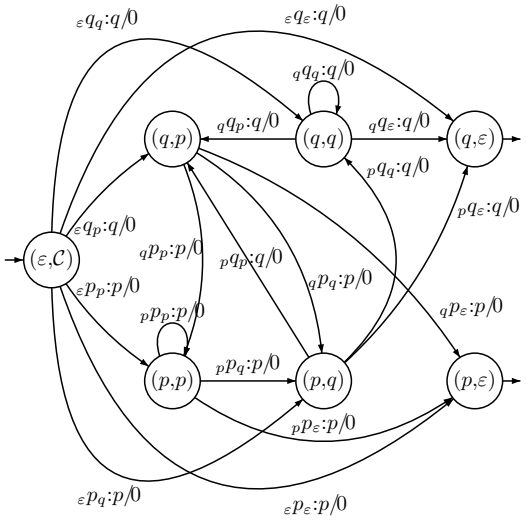


**Figure 4.9.** Context-dependency transducer restricted to two phones $p$ and $q$.

### 4.3.4. Context-dependency transduction

The pronunciation of a phone depends on its neighboring phones. To design an accurate acoustic model, it is thus beneficial to model a *context-dependent phone*, i.e., a phone in the context of its surrounding phones. This has also been corroborated by empirical evidence. The standard models used in speech recognition are $n$-phonic models. A context-dependent phone is then a phone in the context of its $n_1$ previous phones and $n_2$ following phones, with $n_1+n_2+1 = n$. Remarkably, the mapping $\mathbf{P}(c \mid d)$ from phone sequences to sequences of context-dependent phones can be represented by finite-state transducers. This section illustrates that construction in the case of triphonic models ($n_1 = n_2 = 1$). The extension to the general case is straightforward.

Let $\mathcal{P}$ denote the set of context-independent phones and let $\mathcal{C}$ denote the set of triphonic context-dependent phones. For a language such as English or French, $\text{Card}(\mathcal{P}) \approx 50$. Let $_{p_1}p_{p_2}$ denote the context-dependent phone corresponding to the phone $p$ with the left context $p_1$ and the right context $p_2$.

The construction of the context-dependency transducer is similar to that of the language model automaton. As in the previous case, for numerical instability reasons, negative log-probabilities are used, thus the transducer is defined in the log semiring. Each state encodes a history limited to the last two phones. There is a transition from the state associated to $(p, q)$ to $(q, r)$ with input label the context-dependent phone $_pq_r$ and output label $q$. More precisely, the transducer $\mathfrak{T} = (\mathcal{C}, \mathcal{P}, Q, I, F, E, \lambda, \rho)$ is defined by:

- $Q = \{(p, q) : p \in \mathcal{P}, q \in \mathcal{P} \cup \{\varepsilon\}\} \cup \{(\varepsilon, \mathcal{C})\}$;
- $I = \{(\varepsilon, \mathcal{C})\}$ and $F = \{(p, \varepsilon) : p \in \mathcal{P}\}$;

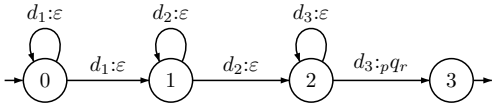


**Figure 4.10.** Hidden-Markov Model transducer.

- $E \subseteq \{((p, Y),\ {}_pq_r, q, 0, (q, r)) : Y = q \text{ or } Y = \mathcal{C}\}$

with all initial and final weights equal to zero. Figure 4.9 shows that transducer in the simple case where the phonemic alphabet is reduced to two phones ($\mathcal{P} = \{p, q\}$). We will denote by $\mathfrak{C}$ the weighted transducer representing the context-dependency mapping.

### 4.3.5. Acoustic model

In most modern speech recognition systems, context-dependent phones are modeled by three-state Hidden Markov Models (HMMs). Figure 4.10 shows the graphical representation of that model for a context-dependent model ${}_pq_r$. The context-dependent phone is modeled by three states (0, 1, and 2) each modeled with a distinct distribution $(d_0, d_1, d_2)$ over the input observations. The mapping $\mathbf{P}(d \mid c)$ from sequences of context-dependent phones to sequences of distributions is the transducer obtained by taking the closure of the union of the finite-state transducers associated to all context-dependent phones. We will denote by $\mathfrak{H}$ that transducer. Each distribution $d_i$ is typically a mixture of Gaussian distributions with mean $\mu$ and covariance matrix $\sigma$:

$$\mathbf{P}(\omega) = \frac{1}{(2\pi)^{N/2}|\sigma|^{1/2}} e^{-\frac{1}{2}(\omega-\mu)^T\sigma^{-1}(\omega-\mu)} \tag{4.3.10}$$

where $\omega$ is an observation vector of dimension $N$. Observation vectors are obtained by local spectral analysis of the speech waveform at regular intervals, typically every 10 ms. In most cases, they are 39-dimensional feature vectors ($N = 39$). The components are the 13 *cepstral coefficients*, i.e., the energy and the first 12 components of the *cepstrum* and their first-order (*delta cepstra*) and second-order differentials (*delta-delta cepstra*). The cepstrum of the (speech) signal is the result of taking the inverse-Fourier transform of the log of its Fourier transform. Thus, if we denote by $x(\omega)$ the Fourier transform of the signal, the 12 first coefficients $c_n$ in the following expression:

$$\log|x(\omega)| = \sum_{n=-\infty}^{\infty} c_n e^{-in\omega} \tag{4.3.11}$$

are the coefficients used in the observation vectors. This truncation of the Fourier transform helps smooth the log magnitude spectrum. Empirically, cepstral coefficients have shown to be excellent features for representing the speech

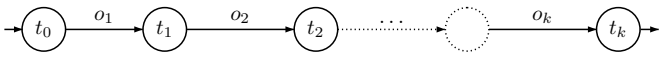


**Figure 4.11.** Observation sequence $O = o_1 \cdots o_k$. The time stamps $t_i$, $i = 0, \ldots k$, labeling states are multiples of 10 ms.

signal.[7] Thus the observation sequence $o = o_1 \cdots o_k$ can be represented by a sequence of 39-dimensional feature vectors extracted from the signal every 10 ms. This can be represented by a simple automaton as shown in figure 4.11, that we will denote by $\mathfrak{O}$.

We will denote by $\mathfrak{O} \star \mathfrak{H}$ the weighted transducer resulting from the application of the transducer $\mathfrak{H}$ to an observation sequence $\mathfrak{O}$. $\mathfrak{O} \star \mathfrak{H}$ is the weighted transducer mapping $O$ to sequences of context-dependent phones, where the weights of the transitions are the negative log of the value associated by a distribution $d_i$ to an observation vector $O_j$, -log $d_i(O_j)$.

### 4.3.6. Combination and search

The previous sections described the representation of each of the components of a speech recognition system by a weighted transducer or weighted automaton. This section shows how these transducers and automata can be combined and searched efficiently using the weighted transducer algorithms previously described, following Equation 4.3.4.

A so-called *Viterbi approximation* is often used in speech recognition. It consists of approximating a sum of probabilities by its dominating term:

$$\hat{w} = \operatorname*{argmax}_w \sum_{d,c,p} \mathbf{P}(o \mid d)\mathbf{P}(d \mid c)\mathbf{P}(c \mid p)\mathbf{P}(p \mid w)\mathbf{P}(w) \tag{4.3.12}$$

$$\approx \operatorname*{argmax}_w \max_{d,c,p} \mathbf{P}(o \mid d)\mathbf{P}(d \mid c)\mathbf{P}(c \mid p)\mathbf{P}(p \mid w)\mathbf{P}(w) \tag{4.3.13}$$

This has been shown to be empirically a relatively good approximation, though, most likely, its introduction was originally motivated by algorithmic efficiency. For numerical instability reasons, negative-log probabilities are used, thus the equation can be reformulated as:

$$\hat{w} = \operatorname*{argmin}_w \min_{d,c,p} -\log \mathbf{P}(o \mid d) - \log \mathbf{P}(d \mid c) - \log \mathbf{P}(c \mid p) - \log \mathbf{P}(p \mid w) - \log \mathbf{P}(w)$$

As discussed in the previous sections, these models can be represented by weighted transducers. Using the composition algorithm for weighted transducers, and by definition of the $\star$-operation and projection, this is equivalent

[7] Most often, the spectrum is first transformed using the Mel Frequency bands, which is a non-linear scale approximating the human perception.

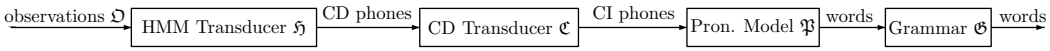


**Figure 4.12.** Cascade of speech recognition transducers.

to:[8]

$$\hat{w} = \operatorname*{argmin}_{w} \Pi_2(\mathfrak{O} \star \mathfrak{H} \circ \mathfrak{C} \circ \mathfrak{P} \circ \mathfrak{G}) \tag{4.3.14}$$

Thus, speech recognition can be formulated as a cascade of composition of weighted transducers illustrated by Figure 4.12. $\hat{w}$ labels the path of $\mathfrak{W} = \Pi_2(\mathfrak{O} \star \mathfrak{H} \circ \mathfrak{C} \circ \mathfrak{P} \circ \mathfrak{G})$ with the lowest weight. The problem can be viewed as a classical single-source shortest-paths algorithm over the weighted automaton $\mathfrak{W}$. Any single-source shortest paths algorithm could be used to solve it. In fact, since $\mathfrak{O}$ is finite, the automaton $\mathfrak{W}$ could be acyclic, in which case the classical linear-time single-source shortest-paths algorithm based on the topological order could be used.

However, this scheme is not practical. This is because the size of $\mathfrak{W}$ can be prohibitively large even for recognizing short utterances. The number of transitions of $\mathfrak{O}$ for 10s of speech is 1000. If the recognition transducer $\mathfrak{T} = \mathfrak{H} \circ \mathfrak{C} \circ \mathfrak{P} \circ \mathfrak{G}$ had in the order of just 100M transitions, the size of $\mathfrak{W}$ would be in the order of $1000 \times 100$M transitions, i.e., about 100 billion transitions!

In practice, instead of visiting all states and transitions, a heuristic pruning is used. A pruning technique often used is the *beam search*. This consists of exploring only states with tentative shortest-distance weights within a *beam* or threshold of the weight of the best *comparable* state. Comparable states must roughly correspond to the same observations, thus states of $\mathfrak{T}$ are visited in the order of analysis of the input observation vectors, i.e. chronologically. This is referred to as a *synchronous beam search*. A synchronous search restricts the choice of the single-source shortest-paths problem or the relaxation of the tentative shortest-distances. The specific single-source shortest paths algorithm then used is known as the *Viterbi Algorithm*, which is presented in Exercise 1.3.1.

The $\star$-operation, the Viterbi algorithm, and the beam pruning techniques are often combined into a *decoder*. Here is a brief description of the decoder. For each observation vector $o_i$ read, the transitions leaving the current states of $\mathfrak{T}$ are expanded, the $\star$-operation is computed on-the-fly to compute the acoustic weights given by the application of the distributions to $o_i$. The acoustic weights are added to the existing weight of the transitions and out of the set of states

[8] Note that the Viterbi approximation can be viewed simply as a change of semiring, from the log semiring to the tropical semiring. This does not affect the topology or the weights of the transducers but only their interpretation or use. Also, note that composition does not make use of the first operation of the semiring, thus compositions in the log and tropical semiring coincide.

reached by these transitions those with a tentative shortest-distance beyond a pre-determined threshold are pruned out. The beam threshold can be used as a means to select a trade-off between recognition speed and accuracy. Note that the pruning technique used is non-admissible. The best overall path may fall out of the beam due to local comparisons.

### 4.3.7. Optimizations

The characteristics of the recognition transducer $\mathfrak{T}$ were left out of the previous discussion. They are however key parameters for the design of real-time large-vocabulary speech recognition systems. The search and decoding speed critically depends on the size of $\mathfrak{T}$ and its non-determinism. This section describes the use of the determinization, minimization, and weight pushing algorithm for constructing and optimizing $\mathfrak{T}$.

The component transducers described can be very large in speech recognition applications. The weighted automata and transducers we used in the North American Business news (NAB) dictation task with a vocabulary of just 40,000 words (the full vocabulary in this task contains about 500,000 words) had the following attributes:

- $\mathfrak{G}$: a *shrunk* Katz back-off trigram model with about 4M transitions;[9]
- $\mathfrak{P}$ : pronunciation transducer with about $70,000$ states and more than 150,000 transitions;
- $\mathfrak{C}$: a triphonic context-dependency transducer with about 1,500 states and 80,000 transitions.
- $\mathfrak{H}$: an HMM transducer with more than 7,000 states.

A full construction of $\mathfrak{T}$ by composition of such transducers without any optimization is not possible even when using very large amounts of memory. Another problem is the non-determinism of $\mathfrak{T}$. Without prior optimization, $\mathfrak{T}$ is highly non-deterministic, thus, a large number of paths need to be explored at the search and decoding time, thereby considerably slowing down recognition.

Weighted determinization and minimization algorithms provide a general solution to both the non-determinism and the size problem. To construct an optimized recognition transducer, weighted transducer determinization and minimization can be used at each step of the composition of each pair of component transducers. The main purpose of the use of determinization is to eliminate non-determinism in the resulting transducer, thereby substantially reducing recognition time. But, its use at intermediate steps of the construction also helps improve the efficiency of composition and reduce the size of the resulting transducer. We will see later that it is in fact possible to construct offline the recognition transducer and that its size is practical for real-time speech recognition!

[9] Various *shrinking methods* can be used to reduce the size of a statistical grammar without affecting its accuracy excessively.

However, as pointed out earlier, not all weighted automata and transducers are determinizable, e.g., the transducer $\mathfrak{P} \circ \mathfrak{G}$ mapping phone sequences to words is in general not determinizable. This is clear in presence of homophones. But even in the absence of homophones, $\mathfrak{P} \circ \mathfrak{G}$ may not have the twins property and be non-determinizable. To make it possible to determinize $\mathfrak{P} \circ \mathfrak{G}$, an auxiliary phone symbol denoted by $\#_0$ marking the end of the phonemic transcription of each word can be introduced. Additional auxiliary symbols $\#_1 \ldots \#_{k-1}$ can be used when necessary to distinguish homophones as in the following example:

| | |
|---|---|
| `r eh d` $\#_0$ | *read* |
| `r eh d` $\#_1$ | *red* |

At most $D$ auxiliary phones, where $D$ is the maximum degree of homophony, are introduced. The pronunciation transducer augmented with these auxiliary symbols is denoted by $\tilde{\mathfrak{P}}$. For consistency, the context-dependency transducer $\mathfrak{C}$ must also accept all paths containing these new symbols. For further determinizations at the context-dependent phone level and distribution level, each auxiliary phone must be mapped to a distinct context-dependent phone. Thus, self-loops are added at each state of $\mathfrak{C}$ mapping each auxiliary phone to a new auxiliary context-dependent phone. The augmented context-dependency transducer is denoted by $\tilde{\mathfrak{C}}$.

Similarly, each auxiliary context-dependent phone must be mapped to a new distinct distribution. $D$ self-loops are added at the initial state of $\mathfrak{H}$ with auxiliary distribution input labels and auxiliary context-dependency output labels to allow for this mapping. The modified HMM transducer is denoted by $\tilde{\mathfrak{H}}$.

It can be shown that the use of the auxiliary symbols guarantees the determinizability of the transducer obtained after each composition. Weighted transducer determinization is used at several steps of the construction. An $n$-gram language model $\mathfrak{G}$ is often constructed directly as a deterministic weighted automaton with a back-off state – in this context, the symbol $\varepsilon$ is treated as a regular symbol for the definition of determinism. If this does not hold, $\mathfrak{G}$ is first determinized. $\tilde{\mathfrak{P}}$ is then composed with $\mathfrak{G}$ and determinized: $det(\tilde{\mathfrak{P}} \circ \mathfrak{G})$. The benefit of this determinization is the reduction of the number of alternative transitions at each state to at most the number of distinct phones at that state ($\approx 50$), while the original transducer may have as many as $V$ outgoing transitions at some states where $V$ is the vocabulary size. For large tasks where the vocabulary size can be more than several hundred thousand, the advantage of this optimization is clear.

The inverse of the context-dependency transducer might not be deterministic.[10] For example, the inverse of the transducer shown in Figure 4.9 is not deterministic since the initial state admits several outgoing transitions with the same input label $p$ or $q$. To construct a small and efficient integrated transducer, it is important to first determinize the inverse of $\mathfrak{C}$.[11]

[10] The inverse of a transducer is the transducer obtained by swapping input and output labels of all transitions.

[11] Triphonic or more generally $n$-phonic context-dependency models can also be constructed directly with a deterministic inverse.

$\tilde{\mathfrak{C}}$ is then composed with the resulting transducer and determinized. Similarly $\tilde{\mathfrak{H}}$ is composed with the context-dependent transducer and determinized. This last determinization increases sharing among HMM models that start with the same distributions: at each state of the resulting integrated transducer, there is at most one outgoing transition labeled with any given distribution name. This leads to a substantial reduction of the recognition time.

As a final step, the auxiliary distribution symbols of the resulting transducer are simply replaced by $\varepsilon$'s. The corresponding operation is denoted by $\Pi_\varepsilon$. The sequence of operations just described is summarized by the following construction formula:

$$\mathfrak{N} = \Pi_\varepsilon(det(\tilde{\mathfrak{H}} \circ det(\tilde{\mathfrak{C}} \circ det(\tilde{\mathfrak{P}} \circ \mathfrak{G})))) \tag{4.3.15}$$

where parentheses indicate the order in which the operations are performed. Once the recognition transducer has been determinized, its size can be further reduced by minimization. The auxiliary symbols are left in place, the minimization algorithm is applied, and then the auxiliary symbols are removed:

$$\mathfrak{N} = \Pi_\varepsilon(min(det(\tilde{\mathfrak{H}} \circ det(\tilde{\mathfrak{C}} \circ det(\tilde{\mathfrak{P}} \circ \mathfrak{G}))))) \tag{4.3.16}$$

Weighted minimization can also be applied after each determinization step. It is particularly beneficial after the first determinization and often leads to a substantial size reduction. Weighted minimization can be used in different semirings. Both minimization in the tropical semiring and minimization in the log semiring can be used in this context. The results of these two minimizations have exactly the same number of states and transitions and only differ in how weight is distributed along paths. The difference in weights arises from differences in the definition of the key pushing operation for different semirings.

Weight pushing in the log semiring has a very large beneficial impact on the pruning efficacy of a standard Viterbi beam search. In contrast, weight pushing in the tropical semiring, which is based on lowest weights between paths described earlier, produces a transducer that may slow down beam-pruned Viterbi decoding many fold.

The use of pushing in the log semiring preserves a desirable property of the language model, namely that the weights of the transitions leaving each state be normalized as in a probabilistic automaton. Experimental results also show that pushing in the log semiring makes pruning more effective. It has been conjectured that this is because the acoustic likelihoods and the transducer probabilities are then synchronized to obtain the optimal likelihood ratio test for deciding whether to prune. It has been further conjectured that this reweighting is the best possible for pruning. A proof of these conjectures will require a careful mathematical analysis of pruning.

The result $\mathfrak{N}$ is an integrated recognition transducer that can be constructed even in very large-vocabulary tasks and leads to a substantial reduction of the recognition time as shown by our experimental results. Speech recognition is thus reduced to the simple Viterbi beam search described in the previous section applied to $\mathfrak{N}$.

In some applications such as for spoken-dialog systems, one may wish to modify the input grammar or language model $\mathfrak{G}$ as the dialog proceeds to exploit the context information provided by previous interactions. This may be to activate or deactivate certain parts of the grammar. For example, after a request for a location, the date sub-grammar can be made inactive to reduce alternatives.

The offline optimization techniques just described can sometimes be extended to the cases where the changes to the grammar $\mathfrak{G}$ are pre-defined and limited. The grammar can then be factored into sub-grammars and an optimized recognition transducer is created for each. When deeper changes are expected to be made to the grammar as the dialog proceeds, each component transducer can still be optimized using determinization and minimization and the recognition transducer $\mathfrak{N}$ can be constructed on-demand using an on-the-fly composition. States and transitions of $\mathfrak{N}$ are then expanded as needed for the recognition of each utterance.

This concludes our presentation of the application of weighted transducer algorithms to speech recognition. There are many other applications of these algorithms in speech recognition, including their use for the optimization of the word or phone lattices output by the recognizer that cannot be covered in this short chapter.

We presented several recent weighted finite-state transducer algorithms and described their application to the design of large-vocabulary speech recognition systems where weighted transducers of several hundred million states and transitions are manipulated. The algorithms described can be used in a variety of other natural language processing applications such as information extraction, machine translation, or speech synthesis to create efficient and complex systems. They can also be applied to other domains such as image processing, optical character recognition, or bioinformatics, where similar statistical models are adopted.

## Notes

Much of the theory of weighted automata and transducers and their mathematical counterparts, rational power series, was developed several decades ago. Excellent reference books for that theory are Eilenberg (1974), Salomaa and Soittola (1978), Berstel and Reutenauer (1984) and Kuich and Salomaa (1986).

Some essential weighted transducer algorithms such as those presented in this chapter, e.g., composition, determinization, and minimization of weighted transducers are more recent and raise new questions, both theoretical and algorithmic. These algorithms can be viewed as the generalization to the weighted case of the composition, determinization, minimization, and pushing algorithms described in Chapter 1 Section 1.5. However, this generalization is not always straightforward and has required a specific study.

The algorithm for the composition of weighted finite-state transducers was given by Pereira and Riley (1997) and Mohri, Pereira, and Riley (1996). The

composition filter described in this chapter can be refined to exploit information about the composition states, e.g., the finality of a state or whether only $\varepsilon$-transitions or only non $\varepsilon$-transitions leave that state, to reduce the number of non-coaccessible states created by composition.

The generic determinization algorithm for weighted automata over weakly left divisible left semirings presented in this chapter as well as the study of the determinizability of weighted automata are from Mohri (1997). The determinization of (unweighted) finite-state transducers can be viewed as a special instance of this algorithm. The definition of the twins property was first formulated for finite-state transducers by Choffrut (see Berstel (1979) for a modern presentation of that work). The generalization to the case of weighted automata over the tropical semiring is from Mohri (1997). A more general definition for a larger class of semirings, including the case of finite-state transducers, as well as efficient algorithms for testing the twins property for weighted automata and transducers under some general conditions is presented by Allauzen and Mohri (2003).

The weight pushing algorithm and the minimization algorithm for weighted automata were introduced by Mohri 1997. The general definition of shortest-distance and that of $k$-closed semirings and the generic shortest-distance algorithm mentioned appeared in Mohri (2002). Efficient implementations of the weighted automata and transducer algorithms described as well as many others are incorporated in a general software library, AT&T FSM Library, whose binary executables are available for download for non-commercial use (Mohri et al. (2000)).

Bahl, Jelinek, and Mercer 1983 gave a clear statistical formulation of speech recognition. An excellent tutorial on Hidden Markov Model and their application to speech recognition was presented by Rabiner (1989). The problem of the estimation of the probability of unseen sequences was originally studied by Good 1953 who gave a brilliant discussion of the problem and provided a principled solution. The back-off $n$-gram statistical modeling is due to Katz (1987). See Lee (1990) for a study of the benefits of the use of context-dependent models in speech recognition.

The use of weighted finite-state transducers representations and algorithms in statistical natural language processing was pioneered by Pereira and Riley (1997) and Mohri (1997). Weighted transducer algorithms, including those described in this chapter, are now widely used for the design of large-vocabulary speech recognition systems. A detailed overview of their use in speech recognition is given by Mohri, Pereira, and Riley (2002). Sproat 1997 and Allauzen, Mohri, and Riley 2004 describe the use of weighted transducer algorithms in the design of modern speech synthesis systems. Weighted transducers are used in a variety of other applications. Their recent use in image processing is described by Culik II and Kari (1997).