Université de Montréal

Weighted Finite-State Transducers in Speech Recognition: A Compaction Algorithm for Non-Determinizable Transducers

par

Shouwen Zhang

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maîtrise ès sciences (M.Sc.)
en informatique

Décembre, 2002

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

Weighted Finite-State Transducers in Speech Recognition: A Compaction Algorithm for Non-Determinizable Transducers

Présenté par :

Shouwen Zhang

A été évalué par un jury composé des personnes suivantes :

Langlais, Philippe
Président-rapporteur

Aïmeur, Esma
Directeur de recherche

Dumouchel, Pierre
Codirecteur

Poulin, Pierre
Membre du jury

Mémoire accepté le : 10 janvier 2003

Résumé

Ma thèse donne un aperçu de l'utilisation des transducteurs à états finis pondérés dans le domaine de la reconnaissance de la parole. La théorie des transducteurs permet une manipulation efficace des modèles de langage humain dans les systèmes de reconnaissance de la parole en représentant, de façon générale et naturelle, les différentes composantes du réseau de connaissances. Ce réseau est construit efficacement grâce aux opérations générales qui permettent de combiner, d'optimiser et d'élaguer des transducteurs. La plupart des opérations applicables aux transducteurs ainsi que leur utilisation en reconnaissance de la parole sont décrites en détail. Ensuite, un algorithme permettant de " compacter " les transducteurs non-déterminisables est développé et testé sur plusieurs transducteurs, montrant ainsi son efficacité à diminuer leur taille.

Mots clés: transducteurs à états finis pondérés, reconnaissance de la parole, réseau de connaissances, compactage, déterminisation

Abstract

My thesis surveys the weighted finite-state transducers (WFSTs) approach to speech recognition. WFSTs provide a powerful method for manipulating models of human language in automatic speech recognition systems due to their common and natural representations for each component of speech recognition network. General transducer operations can combine, optimize, search and prune the recognition network efficiently. The most important transducer operations and their applications in speech recognition are described in detail. Then a transducer compaction algorithm for non-determinizable transducers is developed and some test results show its effectiveness in reducing the size of the transducers.

Keywords: weighted finite-state transducers, speech recognition, recognition network compaction, determinization.

LIST OF FIGURES

Acknowledgement

I would like to express my hearty appreciation to my supervisor Esma Aïmeur for her inspiring guidance toward a research work. Her encouragement and advice are the most valuable experience in my career of study.

I am grateful to my co-supervisor Pierre Dumouchel for his kindness in supporting me to get the opportunity to do my research work in CRIM and interesting me in the area of speech recognition. His support also provided me with the opportunity of preparing my future career.

I am also grateful to Gilles Boulianne, Patrick Cardinal and Jun Qiu for their comments and help in doing my research in CRIM.

Special thanks are due to my wife Chunhong Liu for her love and unconditional support.

Chapter 1

Introduction

Continuous Speech Recognition (CSR) is sufficiently mature that a variety of real world applications are now possible including large vocabulary transcription and interactive spoken dialogue. Speech Recognition has been an active field of study since the beginning of the 50's. Great progress has been made, especially since the 70's, using statistical modeling approaches with Hidden Markov Models (HMMs) and is nowadays regarded as one of the promising technologies of the future.

1.1 Continuous Speech Recognition

Speech recognition systems generally assume that the speech signal is a realization of some message encoded as a sequence of one or more symbols [42]. To recognize the underlying symbol sequence given a spoken utterance, the continuous speech waveform is first converted to a sequence of equally spaced discrete parameter vectors. These speech vectors are then transduced into messages by several stages [42]. Each stage can be represented as a component of speech recognition network. Figure 1.1 illustrates the stages for CSR. The speech vectors are first transduced into phones, the minimal units of speech sound in a language that can serve to distinguish one word from another. The phones are then transduced into syllables, the phonological units which are sometimes thought to interpose between the phones and the word level. After that, words are formed by concatenating the syllables and then the recognized sentences are composed with these words.

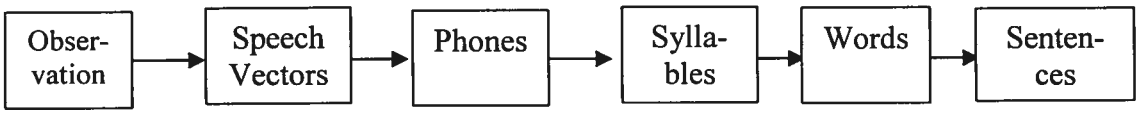


Figure 1.1 Speech Recognition Stages

The statistical approach assumes that the CSR problem is a search problem which is to find the "best" word sentences with the largest probability for a given an utterance. Usually cross-word modeling is used between transduction stages for high-accuracy recognition. In other words, each word can be expanded in a sequence of context-dependent HMM states, conditioned on the neighboring words. The recognized words are then determined by the most probable state sequence.

However, currently the major concerns of CSR are the time and space efficiency, especially for Large Vocabulary Continuous Speech Recognition (LVCSR). Indeed, one of the trends which clearly come out of the new studies of LVCSR is a large increase in the size of data. The effect of the size increase on time and space efficiency is probably the main computational problem one needs to face in LVCSR.

1.2 Finite-State Transducers in Speech Recognition

In the previous section, we have introduced that the spoken utterances are recognized via some transduction stages. Normally, a transduction stage in CSR is modeled by a finite-state device, which is a string-to-string (like the dictionary), string-to-weight (like the language model), or string-to-string/weight transducer (like the hidden Markov models). Each finite-state device in a transduction stage stands for a component of a recognition network.

The application of finite-state transducers in natural language and speech processing is a popular research area [6, 17, 18, 19, 25, 29]. This area is attracting a great deal of attention in the research in Speech Recognition because each component of the recognition network can be represented by transducers (for example, the hidden Markov models) and then these representations can be flexibly and efficiently combined and optimized by transducer operations. The use of finite-state transducers in speech recognition is mainly motivated by considerations of time and space efficiency. Time efficiency is usually achieved by using sequential/deterministic transducers. The output of sequential transducers depends, in general linearly, only on the input size and can therefore be considered as optimal from this point of view. Space efficiency is achieved with transducer minimization algorithms [20] for sequential transducers.

The important research topics on the finite-state transducers are their mathematically well-defined operations that can generalize and efficiently implement the common methods for combining and optimizing probabilistic models in speech processing. Furthermore, new optimization opportunities arise from viewing all symbolic levels of CSR modeling as weighted transducers [22, 29]. Thus, weighted finite-state transducers define a common framework with shared algorithms for the representation and use of the models in speech recognition.

The important finite-state transducer operations are composition, determinization, and minimization. The composition can combine all levels of the CSR network into a single integrated network in a convenient, efficient, and general manner. The determinization algorithms try to construct an equivalent sequential transducer of a weighted transducer. Instead of the original non-sequential transducer, this sequential transducer dramatically increases the searching speed in Speech Recognition process. The running time of sequential transducers for specific input depends linearly only on the size of the input. In most cases the determinization of transducer not only increases time efficiency but also space efficiency. The minimization can reduce the size of the CSR network and thus increase the space efficiency.

1.3 Contribution

The main contribution of my research is the presentation of a transducer compaction operation which can be applied on non-determinizable transducers to reduce the size of the transducers. It is useful in CSR to increase time and space efficiency when the recognition network is represented by weighted finite-state transducers. Moreover, the implementation of the transducer compaction operation we have done in Centre de Recherche Informatique de Montréal (CRIM) will be part of the tools for the speech recognizer of CRIM.

1.4 Thesis Organization

The purpose of this thesis is to survey the weighted transducers in speech recognition and then present a transducer compaction algorithm which can be applied on non-determinizable transducers to reduce their size.

First, in Chapter 2, we give a review on statistical speech recognition and finite-state devices. We describe the CSR problem as a decoding problem to find the word sentences with the largest probability. The decoding is done with appropriate search strategy on the recognition network formed by its components which are an acoustic model, a context-dependency phone model, a pronunciation lexicon/dictionary and a language model in cascade. Then, we give an extended description of some finite-state devices, these are automata, string-to-string transducers, weighted acceptors, and weighted transducers. We first describe the definitions and properties of automata. Next we consider the case of string-to-string transducers. These transducers have been successfully used in the representation of large-scale dictionaries. We describe the theoretical bases for the use of these transducers. In particular, we recall classical theorems and give new ones characterizing these transducers. We then consider the case of sequential weighted acceptors and weighted transducers. These transducers appear very interesting in speech recognition. Language models are represented by weighted acceptors and HMMs are represented by weighted transducers. We give new theorems extending the characterizations known for usual transducers to these transducers. We also characterize the unambiguous transducers admitting determinization.

In Chapter 3, we describe some transducer operations. We briefly describe some basic transducer operations such as union, concatenation, Kleene closure, projection, best path, N-best path, pruning, topological sort, reversal, ε-removal, and inversion. Then we give a detailed description of composition, determinization, and minimization. We first formally define the composition operation. We also describe the composition algorithm for ε-free transducers, then its extension for general case composition is given. Composition operation can construct complex transducers from simpler ones and combine different levels of representation in speech recognition. Furthermore we define

an algorithm for determinizing weighted acceptors, then its extension for determinizing weighted transducers is given, and its correctness is proved. We also briefly describe the minimization of sequential transducers which has a complexity equivalent to that of classical automata minimization.

In Chapter 4, we discuss the application of transducers in speech recognition. Each component of a recognition network can be represented by transducers. Then the transducers in network cascade are combined using the composition operation. Finally the network is optimized via determinization and minimization during composition.

In Chapter 5, we present a transducer compaction algorithm. It can apply on non-determinizable transducers to reduce their size. This operation includes five steps: weight pushing, encoding, determinization, minimization, and decoding. We first give the automata determinization algorithm and then the weight pushing algorithm. The automata determinization algorithm is the classical powerset construction algorithm which can transform any non-deterministic finite automaton (NFA) into an equivalent deterministic finite automaton (DFA). The weight pushing algorithm, which is similar to a generic single source shortest distance algorithm, is to push the weight towards the initial state as much as possible. Then we give the classical automata minimization algorithm which can minimize the size of the automata. The transducer compaction operation is just a combination of these algorithms in appropriate order. At last we describe the applications of the transducer compaction operation in speech recognition.

In Chapter 6, we describe the experimental tests of the transducer compaction algorithm. We test it on some transducers we have in CRIM for building a speech recognizer. Test results show that the transducer compaction operation increases time and space efficiency.

In Chapter 7, we summarize the whole research work with conclusions.

Chapter 2

Fundamentals of Continuous Speech Recognition and Finite-State Transducers

This chapter gives a brief overview of the principles and architecture of modern CSR systems, and describes some finite-state devices such as automata, string-to-string transducers, weighted acceptors, and weighted transducers, in terms of their definitions and properties.

2.1 Continuous Speech Recognition

A major breakthrough in speech recognition technology was made in the 1970's when Jelinek and his colleagues from IBM developed the basic methods of applying the principles of statistical pattern recognition to the problem of speech recognition [8]. Systems based on this statistical framework proved to be superior to the former template and rule based systems.

2.1.1 Framework

The statistical formulation of the CSR problem assumes that a speech signal can be represented by a sequence of acoustic vectors $O = o_1 o_2 ... o_T$ which are equally spaced discrete parameter vectors, and the task of a speech recognizer is to find the most probable utterance (sequence of words) $W = w_1 w_2 ... w_K$ for the given acoustic vectors O. This sequence of acoustic vectors is assumed to form an exact representation of the speech waveform on the basis that for the duration covered by a single vector (around 10 ms) the speech waveform can be reasonably regarded as being quasi-stationary. The specific form of the acoustic vectors is chosen so as to minimize the information lost in the encoding and to provide the best match with the distributional assumptions made by the subsequent acoustic modeling. The CSR problem is then cast as a decoding problem in which we seek the word sequence $\hat{W}$ satisfying:

$$\hat{W} = \underset{W}{\operatorname{argmax}} \quad P(W \mid O) \tag{2.1}$$

Using Bayes' formula,

$$\hat{W} = \arg\max_{W} P(O \mid W)P(W) \tag{2.2}$$

$P(W/O)$ denotes the probability that the words W were spoken, given that the evidence O was observed.

$P(W)$ denotes the probability that the word string W will be uttered.

$P(O/W)$ denotes the probability that when the speaker says W the acoustic O will be observed.

Here $P(O/W)$ is determined by generative model and $P(W)$ is determined by a language model. Most of current research represents $P(O/W)$ as an *acoustic model*. Here we separate *pronunciation model* from acoustic model in the generative model for better representation which we will explain later. The CSR problem is thus reduced to designing and estimating appropriate generative and language models, and finding an acceptable decoding strategy for determining $\hat{W}$.

2.1.2 Generative Model for Speech Recognition

Since the vocabulary of possible words might be very large, the words in W are decomposed into a sequence of basic sounds called *base phone* Q of which there will be around 45 distinct types in English [37]. To allow for the possibility of multiple pronunciations, the likelihood $P(O|W)$ can be computed over multiple pronunciations.

$$P(O|W) = \sum_{Q} P(O|Q)P(Q|W) \tag{2.3}$$

Where

$$P(Q|W) = \prod_{k=1}^{K} P(Q_k|w_k) \tag{2.4}$$

And where $P(Q_k|w_k)$ is the probability that word w_k is pronounced by the base phone sequence $Q_k = q_1^{(k)} q_2^{(k)}$.... In practice, there will only be a very small number of possible Q_k for each w_k making the summation in equation 2.3 easily tractable.

Here $P(O|Q)$ is determined by an acoustic model and $P(Q|W)$ is determined by pronunciation model.

The generative model, $P(O|W)$, is typically decomposed into conditionally-independent mappings between levels:

- Acoustic model $P(O|Q)$: mapping from phone sequences to observation sequences.
- Pronunciation model $P(Q|W)$: mapping from word sequences to phone sequences.

2.1.2.1 Acoustic Model

When the Context-dependent (CD) phone model is used, the computation of $P(O|Q)$ can be decomposed as:

$$P(O|Q) = \sum_{m} P(O|M)P(M|Q) \tag{2.5}$$

Where M represents the CD phone sequences.

$P(O|M)$ is determined by HMMs when CD phone model is considered. Whereas for Context-Independent (CI) phone model, $P(O|Q)$ can sufficiently be determined by HMMs.

i. *Hidden Markov Models (HMMs)*

Each base phone q is represented by a continuous density hidden Markov model (HMM) of the form illustrated in Figure 2.1 with transition parameters $\{a_{ij}\}$ and output observation distributions $\{b_j()\}$. The latter are typically Gaussian and since the dimensionality of the acoustic vectors o_t is relatively high, the covariances are constrained to be diagonal.

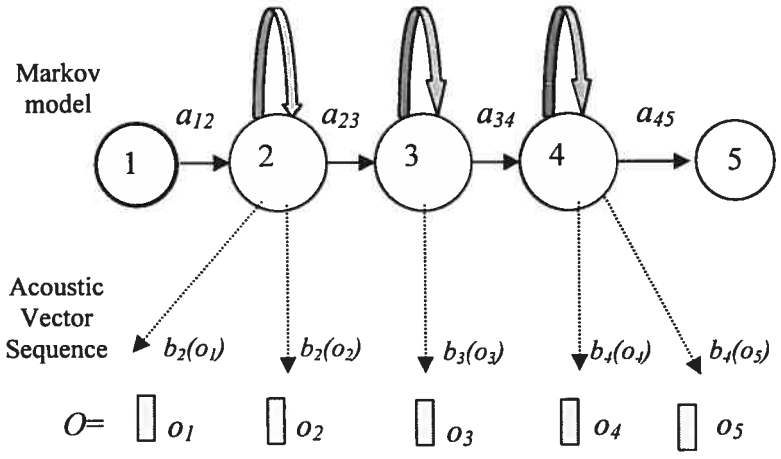


Figure 2.1 HMM with 3 emitting states

Given the composite HMM M formed by concatenating all of the constituent CD model phones the acoustic likelihood is given by

$$P(O|M) = \sum_{X} P(X,O|M) \tag{2.6}$$

where $X = x(0)...x(T)$ is the state sequence through the composite model and

$$p(X, O|d) = a_{x(0),x(1)} \prod_{t=1}^{T} b_{x(t)}(O_t)\, a_{x(t),x(t+1)} \tag{2.7}$$

The acoustic model parameters $\{a_{ij}\}$ and $\{b_j()\}$ can be efficiently estimated from a corpus of training utterances using Expectation-Maximization (EM) which includes a E-step and a M-step [9]. For each utterance, the sequence of baseforms is found and the corresponding composite HMM constructed. A forward-backward alignment is then used to compute state occupation probabilities (the E-step), the means and variances are then maximized via simple weighted averages (the M-step) [9]. Note that in practice the majority of the model parameters are used to model the output distributions and the the transition parameters have little effect on either the likelihood or the recognition accuracy.

ii. *Context-dependent (CD) Phone Models*

$P(M|Q)$ are determined by a CD phone model. $P(M|Q)$ maps the CD model sequences to phone sequences.

In CD phone models each phone is assumed to be able to expand in a sequence of HMM states conditioned on the neighboring phones, usually a triphonic model is used, in which a phone is determined by considering the previous and the following phones. The CD phone models are very useful in high-accuracy speech recognition.

2.1.2.2 Pronunciation Model

$P(Q|W)$ is determined by a pronunciation model which maps the phonemic transcriptions to word sequences. The pronunciation model is usually called *Lexicon* or *pronunciation dictionary.*

2.1.3 Language Model

The probability of a word sequence $W = w_1 w_2 ... w_K$ is

$$P(W) = \prod_{k=1}^{K} P(w_k | w_{k-1}, w_{k-2}, \ldots, w_1) \quad (2.8)$$

For large vocabulary recognition, the conditioning word history in equation 2.7 is usually truncated to *n-1* words to form an *N-Gram* language model

$$P(W) = \prod_{k=1}^{K} P(w_k | w_{k-1}, w_{k-2}, \ldots, w_{k-n+1}) \quad (2.9)$$

Where *n* is typically 2 or 3 and never more than 4. The n-gram probabilities are estimated from training texts by counting n-gram occurrences to form Maximum-Likelihood parameter estimates. The major difficulty of this method is data sparsity which is overcome by a combination of discounting and backing-off [38].

2.1.4 Decoding

A modern speech recognition system thus consists of two stochastic knowledge sources, namely the acoustic model and the language model, a lexicon (which in fact may also be a stochastic model), and a phonetic context-dependent network (in case of cross-word modeling) in the search stage [27]. These components are illustrated as a recognition cascade in Figure 2.2.

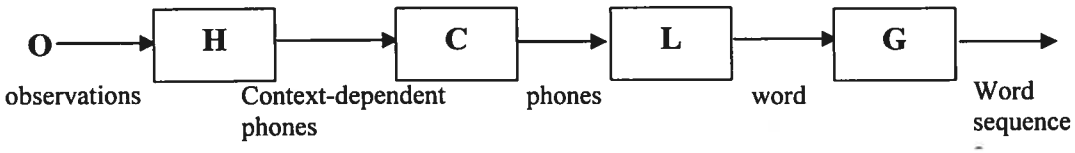


Figure 2.2 Recognition cascade

In Figure 2.2, H is the union of all HMMs used in an acoustic modeling which maps sequences of distribution indices to context-dependent phones. C is a phonetic context-dependent model which maps context-dependent phones to phones for cross-word modeling. L is a pronunciation dictionary or lexicon which maps phonemic transcriptions to word. G is a language model, usually a 3-gram language model is used.

Once the whole speech recognition network is built, Viterbi decoding [42] is usually used to compute the most likely state sequence for an unknown input utterance. Trace back through this sequence then yields the most likely phone and word sequences.

Decoding is a very complex search problem for LVCSR, especially when context-dependent phone model and n-gram language model are used. A standard scheme for reducing search costs uses multiple passes over the data. The output of each pass is a lattice of word sequence hypotheses rather than the single best sequence. This allows the output of one recognition pass to constrain the search in the next pass. An initial pass can use simple models when the search space is large and later passes can use more refined models when the search space is reduced.

However, the weighted transducer approach makes the decoding become easy and fast without resorting to complex schemes. This will be described in detail in the following chapters.

2.2 Finite-State Devices

Finite-state devices, such as finite-state automata, graphs, and finite-state transducers have been known since the emergence of Computer Science and are extensively used in areas such as program compilation, hardware modeling, and database management. Although finite-state devices have been known for a long time, more powerful formalisms such as context-free grammars or unification grammars have been preferred in computational linguistics. However, the richness of the theory of finite-state technology and the mathematical and algorithmic advances resulted in the recent study in the field of finite-state technology have had a great impact on the representation of electronic dictionaries, natural language, and speech processing. As a result, significant developments have been made in many related research areas [7, 18, 26, 29, 34].

Some of the most interesting applications of finite-state machines are concerned with computational linguistics [4, 21, 22, 24, 25]. We can describe these applications from two different views. Linguistically, finite automata are convenient since they allow us to describe easily most of the relevant local phenomena encountered in the empirical study of language by compact representations [5]. Parsing context-free grammars can also be dealt with using finite-state machines, the underlying mechanisms in most of the methods used in parsing are related to automata [6]. From the computational point of view, the use of finite-state machines is mainly motivated by considerations of time and space

efficiency. Actually, the effect of the size increase on time and space efficiency is the main computational problem not only in language and speech processing but also in modern computer science. In language and speech processing, time efficiency is achieved by using deterministic (or sequential) automata. In general, the running time of deterministic finite-state machines for specific input depends linearly only on the size of the input. Space efficiency is achieved with classical minimization algorithms for deterministic automata. Applications such as compiler construction have shown deterministic finite automata to be very efficient [16].

In speech recognition the Weighted Finite-State Transducer (WFST) approach has been considerably studied recently [17, 18, 19, 21, 22, 25, 26, 28, 29]. WFSTs provide a powerful method for manipulating models of human language in automatic speech recognition systems due to their common and natural representations for HMM models, context-dependency, pronunciation dictionaries, grammars, and alternative recognition outputs. Moreover, these representations can be combined by general transducer operations flexibly and efficiently [17, 28]. An efficient recognition network including context-dependent and HMM models can be built using weighted determinization of transducers [18, 22]. The two most important transducer operations, weighted determinization and minimization algorithms, can optimize time and space efficiency. The weights along the paths of a weighted transducer can be distributed optimally by applying a pushing algorithm.

Sequential finite-state transducers are very important devices in natural language and speech processing [18, 19, 20]. Sequential finite-state transducers, simply sequential transducers are also called deterministic transducers. This concept is an extension from deterministic automata to transducers with deterministic inputs. That is a machine which outputs a string or/and weights in addition to accepting (deterministic) inputs.

Hereafter a detailed description of the related finite-state devices used for language processing and speech recognition is given. As basic concepts, we first give an introduction on the definitions and properties of finite-state automata and finite-state

transducers. Then we consider the case of string-to-string transducers, which have been successfully used in the representation of large-scale dictionaries, computational morphology, and local grammars and syntax. Considered next are sequential string-to-weight transducers or so called deterministic weighted acceptors. These transducers are very useful in speech processing. Language models, phoneme lattices, and word lattices are among the objects that can be represented by these transducers.

2.2.1 Finite-State Automata

Finite-State Automata (FSA) can be seen as defining a class of graphs and also as defining languages. The following is a simple description on the definitions of FSA and some closure properties. Other information, such as deterministic FSA (sequential FSA), decidability properties, and space and time efficiency discussion are available from references [4, 31, 35, 36].

2.2.1.1 Definitions

FSA:

A finite-state automaton A is a 5-tuple (Σ, Q, I, F, E), where:

- Σ is a finite set called the alphabet
- Q is a finite set of states
- $I \subseteq Q$ is the set of initial states
- $F \subseteq Q$ is the set of final states
- $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is the set of edges.

By this definition FSA can be seen as a class of graphs.

Figure 2.3 is a finite-state automaton example. It represents a typical left-to-right, three distribution-HMM structure for one phone, with the labels along a complete path specifying legal sequences of acoustic distributions for that phone.

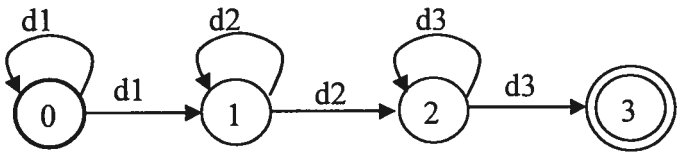


Figure 2.3 A finite-state automaton example

Extended set of edges:

The set of strings built on an alphabet Σ is also called the *free monoid* Σ^*. The formal definition of the star * operation can be found in reference [28]. The extended set of edges $E \subseteq Q \times \Sigma^* \times Q$ is the smallest set such that

(i) $\forall q \in Q, (q, \varepsilon, q) \in E$

(ii) $\forall w \in \Sigma^*$ and $\forall a \in \Sigma \cup \{\varepsilon\}$, if $(q_1, w, q_2) \in E$ and $(q_2, a, q_3) \in E$ then $(q_1, w \bullet a, q_3) \in E$, where $\bullet$ stands for concatenation.

Extended transition function:

The transition function d of a FSA is a mapping from $Q \times (\Sigma \cup \{\varepsilon\})$ to 2^Q, and satisfies $d(q', a) = \{q \in Q | \exists (q', a, q) \in E\}$. The extended transition function d, mapping from $Q \times \Sigma^*$ onto 2^Q, is that function such that

(i) $\forall q \in Q, d(q, \varepsilon) = \{q\}$

(ii) $\forall w \in \Sigma^*$ and $\forall a \in \Sigma \cup \{\varepsilon\}$, $d(q, w \bullet a) = \bigcup_{q_1 \in d(q, w)} d(q_1, a)$

Now, a language $L(A)$ can be defined on finite-state automaton A:

$$L(A) = \{w \in \Sigma^*, i \in I \mid d(i, w) \cap F \neq \varnothing\}$$

A language is said to be regular or recognizable if it can be defined by an FSA.

2.2.1.2 Closure Properties

The set of recognizable language is closed under the following operations:

(1) **Union**. If A_1 and A_2 are two FSAs, it is possible to compute an FSA $A_1 \cup A_2$ such that $L(A_1 \cup A_2) = L(A_1) \cup L(A_2)$.

(2) **Concatenation**. If A_1 and A_2 are two FSAs, it is possible to compute an FSA $A_1 \bullet A_2$ such that $L(A_1 \bullet A_2) = L(A_1) \bullet L(A_2)$.

(3) **Intersection**. If $A_1 = (\Sigma, Q_1, i_1, F_1, E_1)$ and $A_2 = (\Sigma, Q_2, i_2, F_2, E_2)$ are two FSAs, it is possible to compute an FSA denoted $A_1 \cap A_2$ such that $L(A_1 \cap A_2) = L(A_1) \cap L(A_2)$. Such an automaton can be constructed as follows:

$$A_1 \cap A_2 = (\Sigma, Q_1 \times Q_2, (i_1, i_2), F_1 \times F_2, E) \text{ with}$$

$$E = \bigcup_{(q_1,a,r_1)\in E_1,\ (q_2,a,r_2)\in E_2} \{((q_1,q_2),a,(r_1,r_2))\}.$$

(4) **Complementation**. If A is an FSA, it is possible to compute an FSA $-A$ such that $L(-A) = \Sigma^* - L(A)$.

(5) **Kleene Star**. If A is an FSA, it is possible to compute an FSA A^* such that $L(A^*) = L(A)^*$.

2.2.2 Mathematical Foundations for Finite-State Transducers

Before introducing the finite-state transducers we have to give a description about their mathematical foundations. In this section the mathematical foundations such as semiring, power series, weighted transductions, and languages are described in detail.

2.2.2.1 Semiring

The semiring abstraction permits the definition of automata representations and algorithms over a broad class of weight sets and algebraic operations [31]. A semiring $(K, \oplus, \otimes, \bar{0}, \bar{1})$ is a ring that may lack negation. It consists of a set K equiped with an associative and commutative operation $\oplus$, called collection, and an associative operation $\otimes$, called extension, with identities $\bar{0}$ and $\bar{1}$, respectively, such that $\otimes$ distributes over $\oplus$, and $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$. In other words, a semiring is similar to the more familiar ring algebraic structure (such as the ring of polynomial over the reals), except that the additive operation $\oplus$ may not have an inverse. For example, $(N, +, \cdot, 0, 1)$ is a semiring, where 0 and 1 are respectively the identity element for + and $\cdot$ operations with + for collection and $\cdot$ for extension.

The weights used in speech recognition often represent probabilities. The appropriate semiring to use is then the *probability semiring* $(R, +, \cdot, 0, 1)$. However, implementations often replace probabilities with (negative) log probabilities for numerical stability. The appropriate semiring to use is then the image by -log of the semiring $(R, +, \cdot, 0, 1)$ and is called the *log semiring*.

An important example in speech recognition is the *min-sum semiring* or *tropical semiring*, $(R_+ \cup \{\infty\}, \min, +, \infty, 0)$ with min for collection and + for extension. Another *semiring* $(\Sigma^* \cup \{\infty\}, \wedge, \bullet, \infty, \varepsilon)$, called *string semiring*, is also often used in speech recognition, where Σ^*, also called a *free monoid*, defines a set of strings built on an alphabet Σ, $\wedge$ longest common prefix operation, and $\bullet$ concatenation operation, ∞ a new element such that for any string $w \in (\Sigma^* \cup \{\infty\})$ $(w \wedge \infty = \infty \wedge w = w$ and $w \bullet \infty = \infty \bullet w = \infty)$. The cross product of two semirings defines a new semiring.

Some definitions and calculations involve collecting over potentially infinite sets, for instance the set of strings of a language. Clearly, collecting over an infinite set is always well-defined for *idempotent* semirings such as the min-sum semiring, in which $a + a = a$ $\forall a \in K$. More generally, a *closed* semiring is one in which collecting over infinite sets is well-defined.

2.2.2.2 Power Series

A formal power series $S: x \mapsto (S,x)$ is a function from monoid Σ^* to a semiring $(K, \oplus, \otimes, \bar{0}, \bar{1})$. Rational power series are those formal power series that can be built by rational operations (concatenation, sum, and Kleen closure) from the singleton power series given by $(S,x) = k$, $(S,y) = \bar{0}$ if $x \neq y$ for $x \in \Sigma^*$, $k \in K$. The rational power series are exactly those formal power series that can be represented by weighted acceptors which we will discuss later in this chapter. A formal power series S is rational iff it is realizable by a weighted acceptor (recognizable) [9].

2.2.2.3 Weighted Transductions and Languages

A *weighted transduction* T is a mapping $T: \Sigma^* \times \Gamma^* \rightarrow K$ where Σ^* and Γ^* are the sets of strings over the alphabets Σ and Γ, and K is an appropriate weight structure; for instance the real numbers between 0 and 1 in the case of probabilities.

A *weighted language* L is a language satisfying the mapping $L: \Sigma^* \rightarrow K$. Each transduction $S: \Sigma^* \times \Gamma^* \rightarrow K$ has two associated weighted languages, its *first* and *second projections* $\pi_1(S): \Sigma^* \rightarrow K$ and $\pi_2(S): \Gamma^* \rightarrow K$, defined by

$$\pi_1(S)(s) = \Sigma_{t \in \Gamma^*} S(s, t)$$

$$\pi_2(S)(t) = \Sigma_{s \in \Sigma^*} S(s, t)$$

2.2.3 Finite-State Transducers

Finite-state Transducer (FST), also called *string-to-string transducer*, is an extension from FSA. Each arc in FST is labeled by a pair of symbols rather than by a single symbol. A string-to-string transducer is defined on a string semiring.

2.2.3.1 Definitions

FST:

A Finite-State transducer is a 6-tuple $(\Sigma_1, \Sigma_2, Q, i, F, E)$,

where:

- Σ_1 is the input alphabet among a finite set
- Σ_2 is the output alphabet among a finite set
- Q is a finite set of states
- $i \in Q$ is the initial state
- $F \subseteq Q$ is the set of final states
- $E \subseteq Q \times \Sigma_1^* \times \Sigma_2^* \times Q$ is the set of edges

Path:

If an FST $T = (\Sigma_1, \Sigma_2, Q, i, F, E)$, a path of T is a sequence $((p_i,a_i,b_i,q_i))_{i=1,n}$ of edges E such that $q_i = p_{i+1}$ for $i = 1$ to n-1. Where, (p_i,a_i,b_i,q_i) is an edge, q_i is a state which can be reached from state p_i with an input alphabet a_i and an output alphabet b_i.

Successful path:

Given an FST $T = (\Sigma_1, \Sigma_2, Q, i, F, E)$, a successful path $((p_j,a_j,b_j,q_j))_{j=1,n}$ of T is a path of T such that $p_1 = i$ and $q_n \in F$.

These definitions only are part of important definitions on Finite-state transducers and will be frequently used in the following sections. Other definitions and closure properties (for example Union, Inversion, Letter transducer including ε-free transducer and Composition) on FSTs can be found from the literature [12, 26].

2.2.4 Sequential Transducers

Sequential transducers are the most useful transducers used in natural language and speech processing. Many works have been done on this topic [18, 19, 20, 21, 22].

In language and speech processing, sequential transducers are defined as transducers with a deterministic input (string or just a symbol). At any state of such transducers, at most one outgoing arc is labeled with a given element of the alphabet. This means the input is distinct. The output label might be a string (or a single symbol), including the empty string ε. Of course, the output of a sequential transducer is not necessarily deterministic. The formal definition of a sequential string-to-string transducer is as follows:

A sequential transducer is a 7-tuple $(Q, i, F, \Sigma, \Delta, \delta, \sigma)$, where:

- Q is the set of states
- $i \in Q$ is the initial state
- $F \in Q$, the set of final states
- Σ and Δ, finite sets corresponding respectively to the input and output alphabets of the transducer
- δ, the state transition function which maps $Q \times \Sigma$ to Q
- σ, the output function which maps $Q \times \Sigma$ to Δ^*

δ and σ are partial functions (a state $q \in Q$ does not necessarily admit outgoing transitions labeled on the input side with all elements of the alphabet). These functions can be extended to mappings from $Q \times \Sigma^*$ by the following classical recurrence relations:

$$\forall s \in Q, \forall w \in \Sigma^*, \forall a \in \Sigma, \quad \delta(s, \varepsilon) = s, \ \delta(s, wa) = \delta(\delta(s, w), a);$$

$$\sigma(s, \varepsilon) = \varepsilon, \ \sigma(s, wa) = \sigma(s, w)\sigma(\delta(s, w), a).$$

Thus, a string $w \in \Sigma^*$ is accepted by T iff $\delta(i, w) \in F$, and in that case the output of the transducer is $\sigma(i, w)$.

2.2.5 Subsequential and p-Subsequential Transducers

Subsequential transducers are an extension of sequential transducers. By introducing the possibility of generating an additional output string at the final states the application of the transducer to a string can then possibly finish with the concatenation of such an additional output string to the usual output. Such extended sequential transducers with an additional output string at final states are called subsequential transducers.

Language processing often requires a more general extension. Indeed, the ambiguities encountered in language (for example ambiguity of grammars, ambiguity of morphological analyzers, or ambiguity of pronunciation dictionaries) can not be handled by sequential or subsequential transducers because these devices only have a single output to a given input. Since we can not find any reasonable case in language in which the number of ambiguities would be infinite, we can efficiently introduce p-subsequential transducers, namely transducers provided with at most p final output strings at each final state to deal with linguistic ambiguities. However, the number of ambiguities could be very large in some cases. Notice that 1-subsequential transducers are exactly the subsequential transducers. Figure 2.4 shows an example of a 2-subsequential transducer.

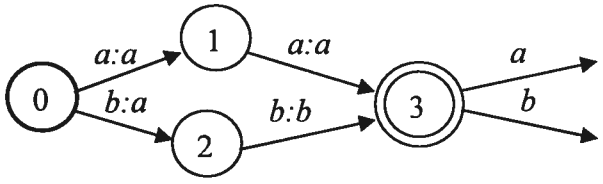


Figure 2.4 A 2-subsequential transducer example

A very important concept here is the sequential/p-subsequential function. Similarly, we define sequential/p-subsequential functions to be those functions that can be represented by sequential/p-subsequential transducers. The following theorems give a brief introduction on the characterizations and properties of subsequential and p-subsequential functions (of course, also that of sequential and p-subsequential transducers). Here, the

expression p-subsequential means two things, the first is that a finite number of ambiguities is admitted, the second indicates that this number equals exactly p.

Theorem 2.2.5.1 (**composition**):

Let $f : \Sigma^* \rightarrow \Delta^*$ be a sequential/p-subsequential and $g : \Delta^* \rightarrow \Omega^*$ be a sequential/q-subsequential function, then $g \circ f$ is sequential/pq-subsequential.

The details about transducer composition are described in Chapter 3.

Theorem 2.2.5.2 (**union**):

Let $f : \Sigma^* \rightarrow \Delta^*$ be a sequential/p-subsequential and $g : \Delta^* \rightarrow \Omega^*$ be a sequential /q-subsequential function, then $g + f$ is 2-subsequential/$(p + q)$-subsequential.

The linear complexity of their use makes sequential and p-subsequential transducers both mathematically and computationally of particular interest. However, not all transducers, even when they realize functions (rational functions), admit an equivalent sequential or subsequential transducer. More generally, sequential functions can be characterized among rational functions by the following theorem.

Theorem 2.2.5.3 (**characterization of sequential function**):

Let f be a rational function mapping Σ^* to Δ^*. f is sequential iff there exists a positive integer K such that:

$$\forall u \in \Sigma^*, \forall a \in \Sigma, \exists w \in \Delta^*, |w| \leq K: f(ua) = f(u)w$$

That is, for any string u and any element a, $f(ua)$ is equal to $f(u)$ concatenated with some bounded string, Notice that this implies that $f(u)$ is always a prefix of $f(ua)$, and more generally that if f is sequential then it preserves prefixes.

The fact that not all rational functions are sequential could reduce the interest of sequential transducers. The following theorem shows however that transducers are exactly compositions of left and right sequential transducers.

Theorem 2.2.5.4 (**composition of left and right sequential transducers**):

Let f be a partial function mapping Σ^* to Δ^*. f is rational iff there exists a left sequential function l: $\Sigma^* \rightarrow \Omega^*$ and a right sequential function r: $\Omega^* \rightarrow \Delta^*$ such that $f = r \circ l$.

Left sequential functions or transducers are those we previously defined. Their application to a string proceeds from left to right. Right sequential functions apply to strings from right to left. According to the theorem, considering a new sufficiently large alphabet Ω allows one to define two sequential functions l and r decomposing a rational function f. This result considerably increases the importance of sequential functions in the theory of finite-state machines as well as in the practical use of transducers.

Sequential transducers offer other theoretical advantages. In particular, while several important tests such as the equivalence are undecidable with general transducers, sequential transducers have the following decidability property.

Theorem 2.2.5.5 (**decidability**):

Let T be a transducer mapping Σ^* to Δ^*. It is decidable whether T is sequential.

The following theorems describe the characterizations of subsequential and p-subsequential functions.

Theorem 2.2.5.6 (**characterization of subsequential function**):

Let f be a partial function mapping Σ^* to Δ^*. F is subsequential iff:

(1) f has bounded variation

(2) for any rational subset Y of Δ^*, $f^{-1}(Y)$ is rational

Theorem 2.2.5.7 (**characterization of *p*-subsequential function**):

Let $f = (f_1, \ldots, f_p)$ be a partial function mapping D o $m(f) \subseteq \Sigma^*$ to $(\Delta^*)^p$. f is p-subsequential iff:

(1) f has bounded variation

(2) for all i ($1 \leq i \leq p$) and any rational subset Y of Δ^*, $f_i^{-1}(Y)$ is rational

Theorem 2.2.5.8 (**characterization of *p*-subsequential function**):

Let f be a rational function mapping Σ^* to $(\Delta^*)^p$. f is p-subsequential iff it has bounded variation.

2.2.6 String-to-Weight Transducers/Weighted Acceptors

A W*eighted Finite-state Acceptor (WFSA),* or a string-to-weight transducer is a finite-state automaton, A, that has both an alphabet symbol and a weight, from some set K, on each transition.

2.2.6.1 Definition and Properties of Weighted Finite-state Acceptors

The definition of WFSA is based on the algebraic structure of a *semiring*, $S=(K, \oplus, \otimes, \bar{0}, \bar{1})$.

Weighted finite-state acceptors or simply weighted acceptors are transducers with input strings and output weights. Figure 2.4 gives an example of a weighted finite-state acceptor, which represents a toy language model.

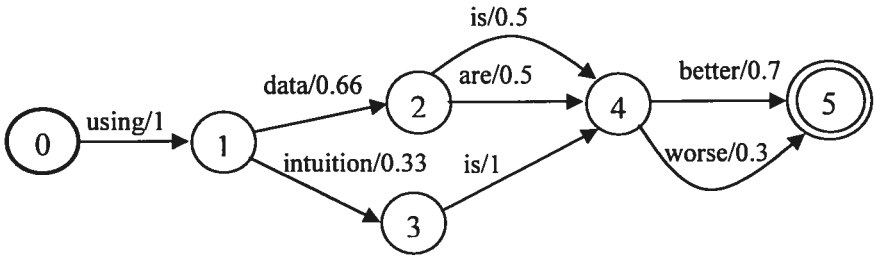


Figure 2.5 A weighted finite-state acceptor

Given a semiring $(K, \oplus, \otimes, \bar{0}, \bar{1})$, the formal definition of a weighted acceptor is as follows:

A weighted acceptor T is defined by $T = (Q, \Sigma, I, F, E, \lambda, \rho)$ over the semiring K, where:

- Q is a finite set of states
- Σ the input alphabet
- $I \subseteq Q$ is the set of initial states
- $F \subseteq Q$, the set of final states
- $E \subseteq Q \times \Sigma \times R_+ \times Q$ a finite set of transitions, where R_+ is the output weight
- λ the initial weight function mapping I to R_+
- ρ the final weight function mapping F to R_+

Compared to the definition of a transducer, we can define for T a partial transition function δ mapping $Q \times \Sigma$ to 2^Q by:

$$\forall (q, a) \in Q \times \Sigma, \delta(q, a) = \{q' \mid \exists x \in R_+ : (q, a, x, q') \in E\},$$

and an output function σ mapping E to R_+ by:

$$\forall t = (p, a, x, q) \in E, \sigma(t) = x.$$

The following concepts and extensions are very important for string-to-weight transducers. Although we have defined some of them in section 3 in general, more details based on string-to-weight transducers are introduced.

A **path** π in T from $q \in Q$ to $q' \in Q$ is a set of successive transitions from q to q': $\pi = ((q_0, a_0, x_0, q_1), \ldots, (q_{m-1}, a_{m-1}, x_{m-1}, q_m))$, with $\forall i \in [0, m\text{-}1]$, $q_{i+1} \in \delta(q_i, a_i)$. We can extend the definition of σ to paths by: $\sigma(\pi) = x_0 x_1 \ldots x_{m-1}$.

The $\pi \in q \sim q' \mid w$ refers to the set of paths from q to q' labeled with the input string w.

The definition of δ can be extended to $Q \times \Sigma^*$ by:

$\forall (q, w) \in Q \times \Sigma^*$, $\delta(q, w) = \{q' : \exists$ path π in $T, \pi \in q \sim q' \mid w\}$ and to $2^Q \times \Sigma^*$, by:

$$\forall R \subseteq Q, \forall w \in \Sigma^*, \delta(R, w) = \bigcup_{q \in R} \delta(q, w).$$

The **minimum of the outputs** of all paths from q to q' labeled with w is defined as:

$$\theta(q, w, q') = \min_{\pi \in (q \sim q') \mid w} \sigma(\pi).$$

A **successful path** in T is a path from an initial state to a final state. A string $w \in \Sigma^*$ is accepted by T iff there exists a successful path labeled with w: $w \in \delta(I, w) \cap F$. The output corresponding to an accepted string w is then obtained by taking the minimum of the outputs of all successful paths with input label w:

$$\min_{(i,f) \in I \times F : f \in \delta(i, w)} (\lambda(i) + \theta(i, w, f) + \rho(f)).$$

A transducer T is said to be **trim** if all states of T belong to a successful path. Weighted acceptors clearly realize functions mapping Σ^* to R_+. Since the operations we need to consider are addition and min, and since $(R_+ \cup \{\infty\}, \min, +, \infty, 0)$ is a semiring, Hence these functions are formal power series. They have the following characterizations which we imported from formal language theory [1, 31]:

1) (S, w) is the image of a string w by a formal power series S. (S, w) is called the coefficient of w in S,
2) by the coefficients, $S = \Sigma_{w \in \Sigma^*} (S, w)w$ can be used to define a power series,
3) the **support** of S is the language defined by:

$$supp(S) = \{w \in \Sigma^* : (S, w) \neq \infty\}.$$

A transducer T is said to be **unambiguous** if for any given string w there exists at most one successful path labeled with w.

2.2.6.2 Sequential String-to-Weight Transducers/Weighted Acceptors

Recall that a transducer is said to be sequential if its input is deterministic, that is, if at any state there exists at most one outgoing transition labeled with a given element of the input alphabet Σ. Sequential weighted acceptors have many advantages over non-sequential weighted acceptors, such as time and space efficiency. But not each weighted acceptor has an equivalent sequential weighted acceptor [18, 29]. The formal definition of a sequential weighted acceptor/string-to-weight transducer is follows:

Definition 2.2.6.1 (**sequential weighted acceptor**):

A sequential weighted acceptor $T = (Q, i, F, \Sigma, \delta, \sigma, \lambda, \rho)$ is an 8-tuple, where:

- Q is the set of its states
- $i \in Q$ its initial state
- $F \subseteq Q$ the set of final states
- Σ the input alphabet
- δ the transition function mapping $Q \times \Sigma$ to Q, δ can be extended as in the string case to map $Q \times \Sigma^*$ to Q
- σ the output function which maps $Q \times \Sigma$ to R_+, σ can also be extended to $Q \times \Sigma^*$
- $\lambda \in R_+$ the initial weight
- ρ the final weight function mapping F to R_+

A string $w \in \Sigma^*$ is accepted by a sequential acceptor T if there exists $f \in F$ such that $\delta(i, w) = f$. Then the output associated to w is: $\lambda + \sigma(i, w) + \rho(f)$.

Considering the benefits of time and space efficiency, the sequential transducer or acceptor is preferred in language and speech processing. But, like we mentioned before, not all transducers are sequential transducers. The process used to transfer a non-sequential transducer to an equivalent sequential transducer is called determinization. Unfortunately, not all transducers have an equivalent sequential transducer, which also

means that not all transducers can be determinized. The following definition can be used to determine whether a transducer can admit determinization.

Definition 2.2.6.2 (**determinization**):

Two states q and q' of a string-to-weight transducer $T = (Q, I, F, \Sigma, \delta, \sigma, \lambda, \rho)$, not necessarily sequential, are said to be **twins** if:

$\forall(u, v) \in (\Sigma^*)^2, (\{q, q'\} \subset \delta(I, u), q \in \delta(q, v), q' \in \delta(q', v)) \Rightarrow \theta(q, v, q) = \theta(q', v, q')$.

If any two states q and q' of a string-to-weight transducer T are twins we say T has **twins property**. If a string-to-weight transducer has twins property it is determinizable. Notice that according to the definition, two states that do not have cycles with the same string v are twins. In particular, two states that do not belong to any cycle are necessarily twins. Thus, an acyclic transducer has the twins property.

The following theorem gives an intrinsic characterization of sequential power series:

Theorem 2.2.6.1 (**characterization of sequential power series**):

Let S be a rational power series defined on the tropical semiring. S is sequential iff it has bounded variation.

The proof on this theorem is based on twins property [3].

2.2.7 Weighted Transducers

Weighted Finite-state Transducers (WFSTs), or simply weighted transducers, are also called string-to-string/weight transducers (SSWTs). The definition of string-to-string/weight transducers is similar to the definition of string-to-string transducers or string-to-weight transducers. The only difference is that the output of a string-to-string/weight transducer is a pair composed by a string and a weight. The SSWTs generalize WFSAs by replacing the single transition label by a pair (i, o) of an input label i and an output label o. While a weighted transducer associates symbol sequences and weights, a WFST associates pairs of symbol sequences and weights, that is, it represents a weighted binary relation between symbol sequences [11, 44].

2.2.7.1 General Weighted Transducers

A formal definition of the weighted transducers is given as the following:

A weighted transducer T is defined by $T = (Q, \Sigma, \Delta, i, F, E, \lambda, \rho)$,

where :

- Q is a finite set of states
- Σ and Δ, finite sets corresponding respectively to the input and output alphabets of the transducer
- $i \in Q$ is the initial state
- $F \subseteq Q$, the set of final states
- $E \subseteq Q \times \Sigma \times \Delta \times R_+ \times Q$ a finite set of transitions
- λ the initial weight function mapping I to R_+
- ρ the final weight function mapping F to R_+

The set E can be extended to include transitions $Q \times \Sigma^* \times \Delta^* \times R_+ \times Q$, where their input and output can be strings.

Without extension of E, it defines the weighted transducers used in our CSR research. Each arc of these transducers has a feature that its input and output are symbols like in Figure 2.5. The symbol refers to a string with a length equals to 1 or 0 (an empty string ε).

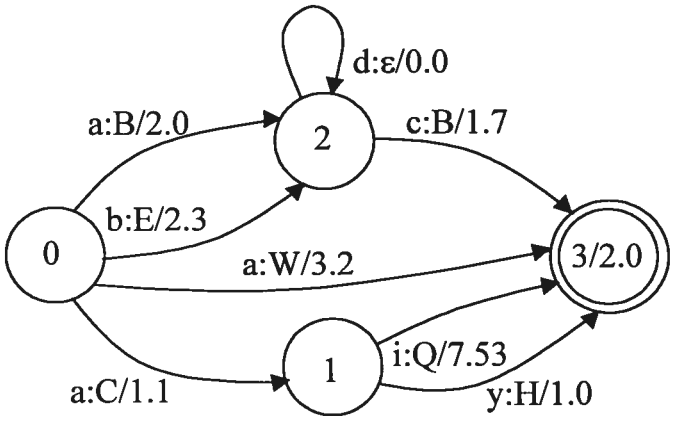


Figure 2.6 A weighted transducer example

2.2.7.2 Sequential Weighted Transducers

The sequential transducers described here are transducers with a deterministic input. At any state of such transducers, at most one outgoing arc is labeled with a given element of the alphabet.

A sequential weighted transducer $T = (Q, i, F, \Sigma, \Delta, \delta, \sigma, \lambda, \rho)$, where :

- Q is the set of its states
- $i \in Q$ its initial state
- $F \subseteq Q$ the set of final states
- Σ and Δ, finite sets corresponding respectively to the input and output alphabets of the transducer
- δ the transition function mapping $Q \times \Sigma$ to Q
- σ the output function which maps $Q \times \Sigma$ to $\Delta \times R_+$
- $\lambda \in R_+$ the initial weight
- ρ the final weight function mapping F to R_+

The transition function δ can be extended as in the string case to map $Q \times \Sigma^*$ to Q, and the output function σ can also be extended to $Q \times \Sigma^*$ to $\Delta^* \times R_+$.

If the extensions of δ and σ are not allowed, the defined sequential weighted transducers will have only symbols as input and output of their arcs like in Figure 2.5. This type of sequential weighted transducers are widely used in current CSR researches.

Even though the sequential property is expected, not all weighted transducers are sequential. In fact, in most cases the original transducer is not sequential. Therefore a determinization algorithm is needed.

In this chapter we have reviewed the principles and architecture of modern CSR systems and then described the formal definitions and properties of finite-state devices including automata, string-to-string transducers, weighted acceptors, and weighted transducers.

In the next chapter we will describe some weighted acceptor and transducer operations.

Chapter 3

Weighted Acceptor and Transducer Operations

Like unweighted acceptors, weighted acceptors and transducers also have a common set of finite-state operations to combine, optimize, search, and prune them [26]. Each operation implements a single, well-defined function that has its foundations in the mathematical theory of rational power series [11, 44]. Many of those operations are the extensions of classical algorithms for unweighted acceptors to weighted transducers.

3.1 Basic Operations

The basic operations, like *union*, *concatenation*, *Kleene closure,* etc., combine transducers in parallel, in series, and with arbitrary repetition, respectively. Other operations include *projection, best path, N-best path, pruning, topological sort, reversal, ε-removal, inversion,* etc. Projection converts transducers to acceptors by projecting onto the input or output label set (projection). Best path or N-best path finds the best or the N best paths in a weighted transducer. Pruning removes unreachable states and transitions. Topological sort operation sorts acyclic automata topologically, that is to number states by satisfying the condition $i \leq j$ for any transition from a state numbered i to a state numbered j. Reversal consists of reversing all transitions of the given transducer, transforming final states into initial states and initial states into final states. ε-removal operation removes all transitions for which the input or output symbols are ε. Inversion operation inverses the transducer by swapping the input with output symbols on transitions.

In following sections a few important operations that support the speech recognition applications are described in detail.

3.2 Composition

The composition operation is the key operation on transducers and is very useful since they allow the construction of more complex transducers from simpler ones.

3.2.1 Theoretical Definition and Operation

Given two transductions T_1: $\Sigma^* \times \Gamma^* \rightarrow K$ and T_2: $\Sigma^* \times \Gamma^* \rightarrow K$, we can define their composition $T_1 \circ T_2$ by

$$(T_1 \circ T_2)(r,t) = \bigoplus_s T_1(r,s) \otimes T_2(s,t)$$

Leaving aside transitions with ε inputs or outputs for the moment, the following rule specifies how to compute a transition of $T_1 \circ T_2$ from appropriate transitions of T_1 and T_2

$$(q_1 \xrightarrow{a:b/w_1} q_1' \text{ and } q_2 \xrightarrow{b:c/w_2} q_2') \Rightarrow (q_1, q_2) \xrightarrow{a:c/(w_1 \otimes w_2)} (q_1', q_2')$$

where $s \xrightarrow{x:y/w} t$ represents a transition from s to t with input x, output y and weight w.

For example, if S represents $P(s_i|s_i)$ and R $P(s_j|s_i)$, $S \circ R$ represents $P(s_j|s_i)$.

It is easy to see that composition $\circ$ is associative, that is, in any transduction cascade $S_1 \circ S_2 \ldots \circ S_n$, the order of association of $\circ$ operations does not matter.

The composition of two transducers represents their relational composition. In particular, the composition $T = R \circ S$ of two transducers R and S has exactly one path mapping sequence u to sequence w for each pair of paths, the first in R mapping u to some sequence v and the second in S mapping v to w. The weight of a path in T is the $\otimes$-product of the weights of the corresponding paths in R and S [11, 44].

Composition is useful for combining different levels of representations. For instance, it can be used to apply a pronunciation lexicon to a word-level grammar to produce a phone-to-word transducer whose word sequences are restricted to the grammar. Many kinds of CSR network combinations, both context-independent and context-dependent, are conveniently and efficiently represented as compositions.

The composition algorithm generalizes the classical state-pair construction for finite automata intersection [18] to weighted acceptors and transducers [7, 29]. The composition $R \circ S$ of transducers R and S has pairs of an R state and an S state as states, and satisfies the following conditions:

(1) its initial state is the pair of the initial states of R and S;

(2) its final states are pairs of a final state of R and a final state of S, and

(3) there is a transition t from (r, s) to (r', s') for each pair of transitions t_R from r to r' and t_S from s to s' such that the output label of t matches the input label of t'. The transition t takes its input label from t_R, its output label from t_S, and its weight is the $\otimes$-product of the weights of t_R and t_S when the weights correspond to probabilities.

Since this computation is *local,* i.e. it involves only the transitions leaving two states being paired and can thus be given a *lazy* implementation in which the composition is generated only as needed by other operations on the composed machine. Transitions with ε labels in R or S must be treated specially as we will discuss later.

Figure 3.1 shows two simple ε-free transducers over the tropical semiring, Figure 3.1a and Figure 3.1b, and the result of their composition, Figure 3.1c. The weight of a path in the resulting transducer is the sum of the weights of the matching paths in R and S since in this semiring, $\otimes$ is defined as the usual addition (of log probabilities).

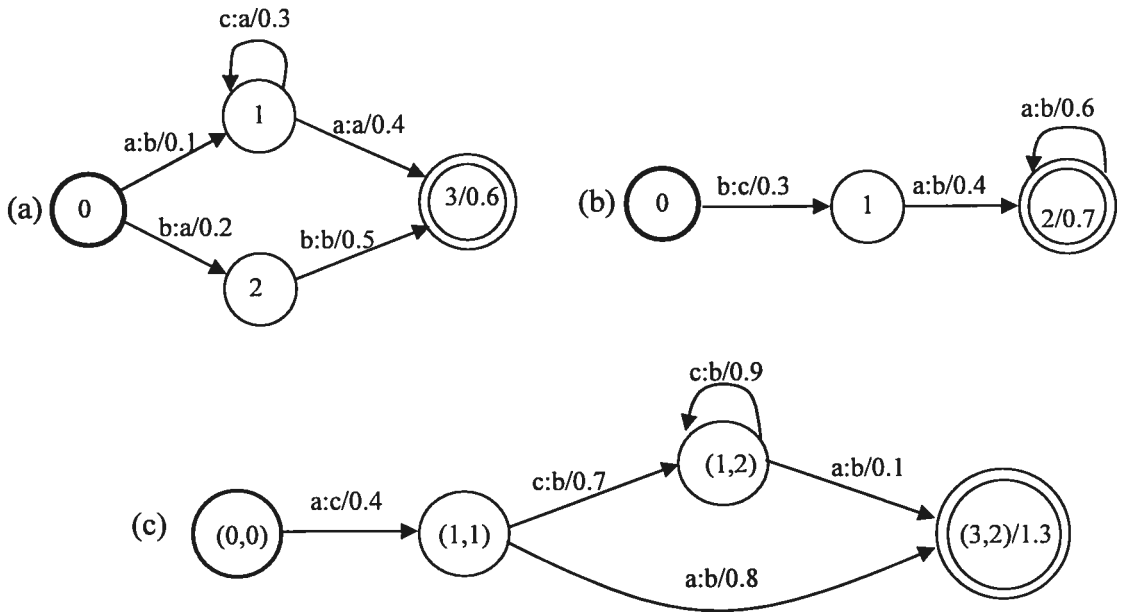


Figure 3.1: Example of ε-free transducer composition

Since weighted acceptors are represented by weighted transducers in which the input and output labels of each transition are identical, the intersection of two weighted acceptors is just the composition of the corresponding transducers.

3.2.2 Composition Algorithm

3.2.2.1 ε *-free composition*

Given two ε-free transducers $T_1 = (Q_1, \Sigma, \Delta, I_1, F_1, E_1, \lambda_1, \rho_1)$ and $T_2 = (Q_2, \Sigma, \Delta, I_2, F_2, E_2, \lambda_2, \rho_2)$, the result of the composition of T_1 and T_2 is transducer $T = (Q, \Sigma, \Delta, i, F, E, \lambda, \rho)$. In this algorithm, transitions are combined using the ⊗-product associated with the semiring over which the transducer is defined. To show the algorithm some notations are introduced :

1. For $q \in Q$, $E[q]$ represents the set of transitions leaving q,
2. For $e \in E$,
   - *i)* $i[e]$ represents the input label of e,
   - *ii)* $o[e]$ represents the output label of e,
   - *iii)* $w[e]$ represents the weight of e,
   - *iv)* $n[e]$ represents the destination state of e.

Now the ε-free composition can be shown as the following pseudocode :

COMPOSITION (T_1, T_2) :

1. $S \leftarrow Q \leftarrow E \leftarrow I \leftarrow F \leftarrow \varnothing$
2. for each $q_1 \in I_1$
3. do for each $q_2 \in I_2$
4. do $Q \leftarrow Q \cup \{(q_1, q_2)\}$
5. $I \leftarrow I \cup \{(q_1, q_2)\}$
6. $\lambda((q_1, q_2)) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_1)$
7. ENQUEUE (S, (q_1, q_2))
8. if $q_1 \in F_1$ and $q_2 \in F_2$
9. then $F \leftarrow F \cup \{(q_1, q_2)\}$

10. $\rho((q_1, q_2)) \leftarrow \rho_1(q_1) \otimes \rho_2(q_2)$

11. while $S \neq \varnothing$

12. do $(q_1, q_2) \leftarrow head[S]$

13. for each $(e_1, e_2) \in E[q_1] \times E[q_2]$ such that $o[e_1] = i[e_2]$

14. do if $(n[e_1], n[e_2]) \notin Q$

15. then $Q \leftarrow Q \cup \{(n[e_1], n[e_2])\}$

16. ENQUEUE $(S, (n[e_1], n[e_2]))$

17. if $n[e_1] \in F_1$ and $n[e_2] \in F_2$

18. then $F \leftarrow F \cup \{(n[e_1], n[e_2])\}$

19. $\rho((n[e_1], n[e_2])) \leftarrow \rho_1(q_1) \otimes \rho_2(q_2)$

20. $E \leftarrow E \cup \{((q_1, q_2), i[e_1], o[e_2], w[e_1] \otimes w[e_2], (n[e_1], n[e_2])\}$

21. ENQUEUE(S)

Figure 3.2 Pseudocode of the ε-free composition

3.2.2.2 General Case Composition

Transitions with ε labels in T_1 or T_2 add some subtleties to composition. In general, output and input ε's can be aligned in several different ways: an output ε in T_1 can be consumed either by staying in the same state in T_2 or by pairing it with an input ε in T_2; an input ε in T_2 can be handled similarly. For instance, the two transducers in Figure 3.3(a) and (b) can generate all the alternative paths in Figure 3.4. However, the single bold path is sufficient to represent the composition result, shown separately in Figure 3.5. The problem with redundant paths is not only that they increase unnecessarily the size of the result, but also they fail to preserve *path multiplicity*: each pair of compatible paths in T_1 and T_2 may yield several paths in $T_1 \circ T_2$. If the weight semiring is not idempotent, that leads to a result that does not satisfy the algebraic definition of composition.

This path-multiplicity problem can be solved by mapping the given composition into a new composition

$$T_1 \circ T_2 \rightarrow T_1' \circ F \circ T_2'$$

in which F is a special *filter* transducer and the T_i' are versions of the T_i in which the relevant ε labels are replaced by special "silent transition" symbols ε_i as shown in Figure

3.3(c) and (d). The bold path in Figure 3.4 is the only one allowed by the filter in Figure 3.6 for the input transducers in Figure 3.3.

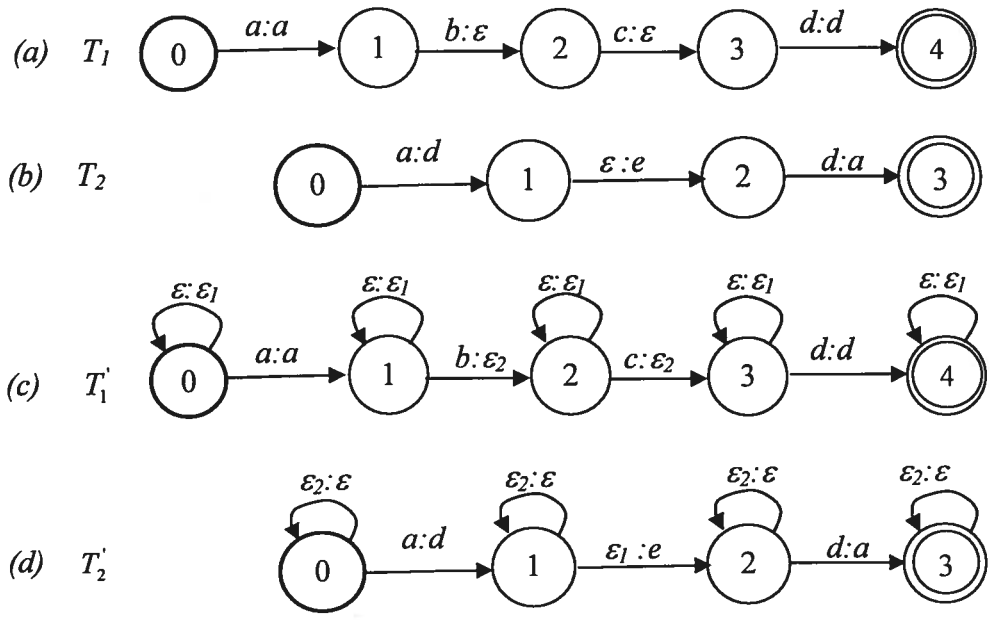


Figure 3.3 Transducers with ε labels

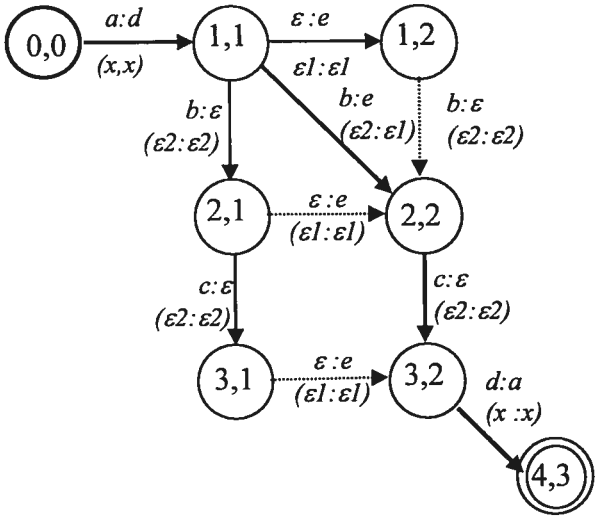


Figure 3.4 Composition with marked ε's

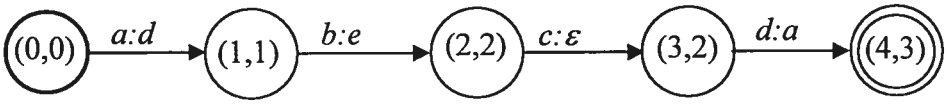


Figure 3.5 Composition output

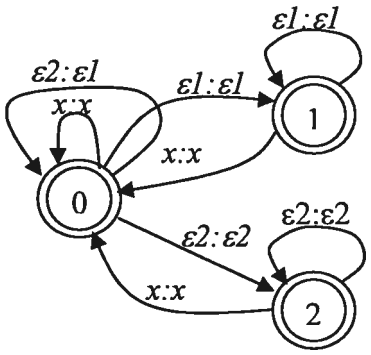


Figure 3.6 Composition filter

By inserting a filter between T_1 and T_2 (more precisely, between T_1' and T_2') and applying the ε-free composition algorithm on this new composition, the redundant paths are removed. Interestingly, the filter itself can be represented as a finite-state transducer. Filters of different forms are possible, but the one shown in Figure 3.6 leads in many cases to the fewest transitions in the result, and often to better time efficiency [25]. (The symbol x represents any element of the alphabet of the two transducers.)

The filter can be understood in the following way: as long as the output of T_1 matches the input of T_2, one can move forward on both and stay at state 0. If there is an ε-transition in T_1 , one can move forward in T_1 (only) and then repeat this operation (state 1) until a possible match occurs which would lead to the state 0 again. Similarly, if there is an ε-transition in T_2, one can move forward in T_2 (only) and then repeat this operation (state 2) until a possible match occurs which would lead to the state 0.

Clearly, all the operations involved in the filtered composition are also local, therefore they can be performed on demand, without the need to perform explicitly the replacement of T_i by T_i'.

We can thus use the lazy composition algorithm as a subroutine in a standard Viterbi decoder to combine on-the-fly a language model, a multi-pronunciation lexicon with corpus-derived pronunciation probabilities, and a context-dependency transducer. The external interface to composed transducers does not distinguish between lazy and precomputed compositions, so the decoder algorithm is the same as for an explicit network.

3.2.2.3 Complexity

In the worst case, the composed transducer results in the combination of all state-pairs (q_1, q_2) as its states and has at most $|E_1|\cdot|E_2|$ transitions. Thus it takes $O(|Q_1|\cdot|Q_2|)$ for the creation of states and $O(|E_1|\cdot|E_2|)$ for the creation of all transitions. Therefore, the overall complexity is:

$$O(|Q_1|\cdot|Q_2| + |E_1|\cdot|E_2|)$$

3.3 Determinization

A deterministic automaton is non-redundant and contains at most one path matching any input sequence, thus reducing time and space required to process an input sequence.

In the same way, a deterministic/sequential WFSA needs to eliminate redundancy. Thus it must calculate the combined weight of all the paths for a given input sequence. For instance, in the case, common in speech recognition, where weights are interpreted as (negative) logarithms of probabilities, the weight of a path is obtained by adding the weights of its transitions, and the combined weight for an input string is the minimum of the weights of all paths accepting that string. In the case where weights are probabilities where the overall probability mass of all paths accepting an input is sought, weights are multiplied along a path and summed across paths. Both cases can be handled by the same algorithm, parameterized with appropriate definitions of the two weight combination operations.

As aforementioned, not all transducers have an equivalent sequential transducer, which means that not all transducers can be determinized. The following definition can be used to determine whether a transducer can admit determinization.

Two states q and q' of a string-to-weight transducer $T = (Q, I, F, \Sigma, \delta, \sigma, \lambda, \rho)$, not necessarily sequential, are said to be *twins* if:

$$\forall(u, v) \in (\Sigma^*)^2, (\{q, q'\} \subset \delta(I, u), q \in \delta(q, v), q' \in \delta(q', v)) \Rightarrow \theta(q, v, q) = \theta(q', v, q').$$

If any two states q and q' of a string-to-weight transducer T are twins, we say T has *twins property*. If a string-to-weight transducer has twins property it is determinizable.

Notice that according to the definition, two states that do not have cycles with the same string v are twins. In particular, two states that do not belong to any cycle are necessarily twins. Thus, an acyclic transducer has the twins property.

The sequential power series in the tropical semiring are functions that can be realized by sequential string-to-weight transducers. Many rational power series defined on the tropical semiring considered in practice are sequential, in particular acyclic transducers represent subsequential power series.

The following theorem gives an intrinsic characterization of sequential power series:

Let S be a rational power series defined on the tropical semiring. S is sequential iff it has bounded variation [18].

3.3.1 Determinization Algorithm for Power Series

In speech recognition systems the tropical semiring is widely used. The determinization algorithm will be frequently applied to the power series defined on tropical semiring. Therefore, the following algorithm is presented in the case of a tropical semiring $(R_+ \cup \{\infty\}, \min, +, \infty, 0)$ on which the transducer is defined. This algorithm is easily changed to fit other semirings by replacing **min** and + by their own binary operations.

The following determinization algorithm constructs an equivalent sequential weighted acceptor $T_2 = (Q_2, i_2, F_2, \Sigma, \delta_2, \sigma_2, \lambda_2, \rho_2)$ to a given non-sequential one $T_1 = (Q_1, \Sigma, I_1, F_1, E_1, \lambda_1, \rho_1)$ defined on tropical semiring [3,10,22].

PowerSeriesDeterminization(T_1, T_2)

1 $F_2 \leftarrow \varnothing$

2 $\lambda_2 \leftarrow \min_{i \in I_1} \lambda_1(i)$

3 $i_2 \leftarrow \bigcup_{i \in I_1} \{(i, \lambda_2^{-1} + \lambda_1(i))\}$

4 $Q \leftarrow \{i_2\}$

5 **while** $Q \neq \varnothing$

6 **do** $q_2 \leftarrow$ head$[Q]$

7 **if** (there exists $(q, x) \in q_2$ such that $q \in F_1$)

8 **then** $F_2 \leftarrow F_2 \cup \{q_2\}$

9 $\rho_2(q_2) \leftarrow \min_{q \in F_1,\ (q,\ x) \in q_2} (x + \rho_1(q))$

10 **for** each a such that $\Gamma(q_2, a) \neq \varnothing$

11 **do** $\sigma_2(q_2, a) \leftarrow \min_{(q,\ x) \in \Gamma(q_2,\ a)} [x + \min_{t=(q,\ a,\ \sigma_1(t),\ n_1(t)) \in E_1} \sigma_1(t)]$

12 $\delta_2(q_2, a) \leftarrow \bigcup_{q' \in \nu(q_2,\ a)} \{(q', \min_{(q,\ x,\ t) \in \gamma(q_2,\ a),\ n_1(t) = q'} ([\sigma_2(q_2, a)]^{-1} + x + \sigma_1(t))\}$

13 **if** ($\delta_2(q_2, a)$ is a new state)

14 **then ENQUEUE**(Q, $\delta_2(q_2, a)$)

15 **DEQUEUE**(Q)

Figure 3.7 Algorithm for the determinization of a weighted acceptor T_1 defined on the semiring $(R_+ \cup \{\infty\}, \min, +, \infty, 0)$

The key points in this algorithm are further explained as follows:

1. Line 2 and line 3 tell us that the initial weight λ_2 of T_2 is the minimum of all the initial weights of T_1. The initial state i_2 is a subset made of pairs (i, x), where i is an initial state of T_1, and $x = \lambda_1(i) - \lambda_2$. We use a queue Q to maintain the set of subsets q_2 not be examined. Initially, Q contains only the subset i_2. The subsets q_2 are the states of the resulting transducer. Q_2 is a final state of T_2 iff it contains at least one pair (q, x), with q a final state of T_1 (lines 7-8). The final output associated to q_2 is then the minimum of the final outputs of all the final states in q_2 combined with their respective residual weights (line 9).

2. For each input label a such that there exists at least one state q of the subset q_2 admitting an outgoing transition labeled with a, one outgoing transition leaving q_2 with the input label a is constructed (lines 10-14). The output $\sigma_2(q_2, a)$ of this transition is the minimum of the outputs of all the transitions with input label a that leave a state in the subset q_2, when combined with the residual weight associated to that state (line 11).

3. The destination $\delta_2(q_2, a)$ of the transition leaving q_2 is a subset made of pairs (q', x'). where q' is a state of T_1 that can be reached by a transition labeled with a, and x' the corresponding residual weight. In line 12, x' is not shown explicitly, but it is computed by taking the minimum of all the transitions with input label a that leave a state q of q_2 and reach q', when combined with the residual weight of q minus the output weight $\sigma_2(q_2, a)$. Finally, $\delta_2(q_2, a)$ is enqueued in Q iff it is a new subset.

4. $n_1(t)$ is the destination state of a transition $t \in E_1$. Hence, $n_1(t) = q'$, if $t = (q, a, x, q') \in E_1$. The sets $\Gamma(q_2, a)$, $\gamma(q_2, a)$ and $\nu(q_2, a)$ used in the algorithm are defined by:

$$\Gamma(q_2, a) = \{(q, x) \in q_2 : \exists t = (q, a, \sigma_1(t), n_1(t)) \in E_1\}$$

$$\gamma(q_2, a) = \{(q, x, t) \in q_2 \times E_1 : t = (q, a, \sigma_1(t), n_1(t)) \in E_1\}$$

$$\nu(q_2, a) = \{(q' : \exists (q, x) \in q_2 : \exists t = (q, a, \sigma_1(t), q') \in E_1\}$$

$\Gamma(q_2, a)$ denotes the set of pairs (q, x), elements of the subset q_2, having transitions labeled with the input a. $\gamma(q_2, a)$ denotes the set of triples (q, x, t) where (q, x) is a pair in q_2 such that q admits a transition with input label a. $\nu(q_2, a)$ is the set of states q' that can be reached by transitions labeled with a from the states of the subset q_2.

Notice that several transitions might reach the same state with different residual weights. Since we are only interested in the best path, namely the path corresponding to the minimum weight, we can keep the minimum of these weights for a given state element of a state (line 11 of the algorithm).

The complexity (both space and time) of this power series determinization algorithm is exponential. However, in some cases in which the degree of nondeterminism of the initial transducer is high, the determinization algorithm turns out to be fast and the resulting transducer has less states.

It has been proved that if the determinization algorithm terminates, then the resulting transducer T_2 is equivalent to T_1.

This power series determinization algorithm is applied to a tropical semiring, i.e. the weighted acceptor. Actually, it is possible to extend the determinization algorithms for different semirings based on this PowerSeriesDeterminization algorithm. For weighted transducers, subsets in the algorithm are made of triples (q, w, x) where q is a state of the original transducer, w is a residual string and x is a residual weight. So, we have to consider the pair (w, x) as output in the determinization of weighted transducers. Also some special cases have to be handled. In fact, based on this general algorithm an efficient determinization algorithm has been developed to deal with the weighted transducers, named *WT_determinization Algorithm* [14].

The determinizable transducers can be simply defined as those transducers with which the determinization algorithm terminates. If a transducer is not determinizable the algorithm will keep running until resources are used up. It has been declared early in this thesis that the complexity of the application of sequential transducers is linear in the size of the string to which it applies. This property makes it worthwhile to use the power series determinization in order to speed up the application of transducers. Unfortunately, not all transducers can be determinized using the power series determinization because determinization does not apply to all transducers. Therefore it is important to be able to test the determinizability of a transducer.

We know that if a transducer defined on the tropical semiring has the twins property then it is determinizable. There are transducers that do not have the twins property and that are still determinizable. Normally it is not an easy job to characterize such transducers because we need more complex conditions [19].

Actually, if we wish to construct the result of the determinization of transducer T for a given input string w, we do not need to expand the whole result of the determinization, but only the necessary part of the determinized transducer. When restricted to a finite set the function realized by any transducer is sequentiable since it has bounded variation. Acyclic transducers have the twins property, so they are determinizable. Therefore, it is

always possible to expand the result of the determinization algorithm for a finite set of input strings, even if T is not determinizable [22].

3.3.2 Weighted Transducer Determinization Algorithm

We have introduced a general *PowerSeriesDeterminization* algorithm, which is applied to a tropical semiring such as a weighted acceptor. This algorithm can be also used to develop determinization algorithms for other semirings.

Here we consider the string semiring $(\Sigma^* \cup \{\infty\}, \wedge, \bullet, \infty, \varepsilon)$, where $\wedge$ denotes the longest common prefix operation. The cross product of two semirings defines a semiring. The general algorithm also applies when the semiring is the cross product of $(\Sigma^* \cup \{\infty\}, \wedge, \bullet, \infty, \varepsilon)$ and $(R_+ \cup \{\infty\}, \min, +, \infty, 0)$. This allows us to determinize transducers outputting pairs of strings and weights -- the weighted transducers.

For the string-to-string/weight transducers, subsets in the algorithm are made of triples $(q, w, x) \in Q \times \Sigma^* \cup \{\infty\} \times R_+ \cup \{\infty\}$ where q is a state of the original transducer, w is a residual string, and x is a residual weight. So, we have to consider the pair (w, x) as output in the determinization of weighted transducers. Also, in order to consider this *PowerSeries-Determinization* algorithm as general, some special cases have to be handled.

Based on the general algorithm and the features of the weighted transducers, a new determinization algorithm called *WT_ determinization* used for weighted transducers has been developed [14, 29].

Figure 3.5 gives the detailed pseudocode of the *WT_determinization* algorithm. This algorithm constructs a sequential weighted transducer $T_2 = (Q_2, i_2, F_2, \Sigma, \Delta, \delta_2, \sigma_2, \lambda_2, \rho_2)$ equivalent to a given determinizable weighted transducer $T_1 = (Q_1, \Sigma, \Delta, I_1, F_1, E_1, \lambda_1, \rho_1)$.

3.3.2.1 Pseudocode and Description of the WT_determinization Algorithm

WT_determinization(T_1, T_2)

1 $F_2 \leftarrow \varnothing$

2 $\lambda_2 \leftarrow \min_{i \in I_1} \lambda_1(i)$

3 $i_2 \leftarrow \bigcup_{i \in I_1} \{(i, \varepsilon, \lambda_2^{-1} + \lambda_1(i))\}$

4 $Q \leftarrow \{i_2\}$

5 **while** $Q \neq \varnothing$

6 **do** $q_2 \leftarrow$ head[Q]

7 **if** ($q \neq \{-2\}$ and for any $(q, w, x) \in q_2$)

8 **if** (there exists $(q, \varepsilon, x) \in q_2$ such that $q \in F_1$ or $q = \{-1\}$)

9 **then** $F_2 \leftarrow F_2 \cup \{q_2\}$

10 $\rho_2(q_2) \leftarrow \min_{q \in F_1 \cup \{-1\},\ (q,\ \varepsilon,\ x) \in q_2,\ \rho_1(\{-1\})=0} (x + \rho_1(q))$

11 **if** (there exists $(q, w, x) \in q_2$ such that $w \neq \varepsilon$, $q \in F_1$ or $q = \{-1\}$)

12 **then** $q_2' \leftarrow \bigcup_{(q,\ w,\ x) \in q_2,\ q \in F_1 \cup \{-1\},\ \rho_1(-1)=0,\ w \neq \varepsilon} (q, w, x + \rho_1(q))$

13 $w' \leftarrow \bigcup_{(q,\ w,\ x) \in q_2'} \mathit{firstSymbol}(w)$

14 **for** each symbol $s \in w'$

15 **do** $\sigma_2(q_2, \varepsilon) \leftarrow (s, \min_{(q,\ w,\ x) \in q_2',\ \mathit{firstSymbol}(w)=s} [x + \rho_1(q)])$

16 $\delta_2(q_2, \varepsilon) \leftarrow \bigcup_{(q,\ w,\ x) \in q_2',\ \mathit{firstSymbol}(w)=s} (\{-1\}, w \bullet (\sigma_2(q_2, \varepsilon)|w)^{-1},$

17 $(\sigma_2(q_2, \varepsilon)|x)^{-1} + x + \rho_1(q))$

18 **if** ($\delta_2(q_2, \varepsilon)$ is a new state)

19 **then ENQUEUE**(Q, $\delta_2(q_2, \varepsilon)$)

20 **for** each a such that $\Gamma(q_2, a) \neq \varnothing$

21 **do** $\sigma_2'(q_2, a) \leftarrow (\bigwedge_{(q, w, x) \in \Gamma(q_2, a)} [w \bullet (\sigma_1(t)|w)],$

$\min_{(q,\ w,\ x) \in \Gamma(q_2,\ a)} [x + \min_{t=(q,\ a,\ \sigma_1(t),\ n_1(t)) \in E_1} \sigma_1(t)|x])$

22 $\delta_2'(q_2, a) \leftarrow \bigcup_{q' \in \nu(q_2,\ a)} \{(q',\ w \bullet (\sigma_1(t)|_w) \bullet (\sigma_2'(q_2, a)|_w)^{-1},$

$\min_{(q,\ w,\ x,\ t) \in \gamma(q_2,\ a),\ w=w_0,\ \sigma_1(t)|_w=w_1,\ n_1(t)=q'} [(\sigma_2'(q_2, a)|_x)^{-1} + x + \sigma_1(t)|_x])\}$

23 **if** ($\sigma_2'(q_2, a)|_w$ is not a symbol and also not a empty string)

24 $w'' \leftarrow \sigma_2'(q_2, a)|_w$

25 $\sigma_2(q_2, a) \leftarrow (firstSymbol(w''),\ \sigma_2'(q_2, a)|_x)$

26 $w'' \leftarrow removeFirstSymbol(w'')$

27 $\delta_2(q_2, a) \leftarrow (\{-2\}, w'', 0) \cup \delta_2'(q_2, a)$

28 **if** ($\delta_2(q_2, a)$ is a new state)

29 **then ENQUEUE**(Q, $\delta_2(q_2, a)$)

30 **else** $\sigma_2(q_2, a) \leftarrow \sigma_2'(q_2, a)$

31 $\delta_2(q_2, a) \leftarrow \delta_2'(q_2, a)$

32 **if** ($\delta_2(q_2, a)$ is a new state)

33 **then ENQUEUE**(Q, $\delta_2(q_2, a)$)

34 **else if**($q = \{-2\}$)

35 $\sigma_2(q_2, \varepsilon) \leftarrow (firstSymbol(w''),\ 0)$

36 $w'' \leftarrow removeFirstSymbol(w'')$

37 **if**($w'' == \varepsilon$)

38 **then** $\delta_2(q_2, \varepsilon) \leftarrow \delta'_2(q_2, a)$

39 **else** $\delta_2(q_2, \varepsilon) \leftarrow (\{-2\}, w'', 0) \cup \delta'_2(q_2, a)$

40 **if**($\delta_2(q_2, \varepsilon)$ is a new state)

41 **then ENQUEUE**(Q, $\delta_2(q_2, \varepsilon)$)

42 **DEQUEUE**(Q)

Figure 3.8 Algorithm for the determinization of a weighted transducer T_1 defined on the semiring $(\Sigma \cup \{\infty\}, \wedge, \bullet, \infty, \varepsilon) \times (R_+ \cup \{\infty\}, \min, +, \infty, 0)$.

This algorithm considers two basic requirements on the resulting transducer T_2. First, the original transducer T_1 used in our CSR research has each its arc with a format *symbol: symbol/weight*. Second, each final state of the original transducer T_1 has only an

accepting weight (or output weight). These two characteristics are kept in the resulting transducer T_2.

In this algorithm, $n_1(t)$ is defined as the destination state of a transition $t \in E_1$. Hence, $n_1(t) = q'$, if $t = (q, a, w, x, q') \in E_1$. The sets $\Gamma(q_2, a)$, $\gamma(q_2, a)$, and $\nu(q_2, a)$ used in the algorithm are defined by:

$$\Gamma(q_2, a) = \{(q, w, x) \in q_2 : \exists t = (q, a, \sigma_1(t), n_1(t)) \in E_1\}$$

$$\gamma(q_2, a) = \{(q, w, x, t) \in q_2 \times E_1 : t = (q, a, \sigma_1(t), n_1(t)) \in E_1\}$$

$$\nu(q_2, a) = \{(q' : \exists (q, w, x) \in q_2 : \exists t = (q, a, \sigma_1(t), q') \in E_1\}$$

$\Gamma(q_2, a)$ denotes the set of triples (q, w, x), elements of the subset q_2, having transitions labeled with the input a. $\gamma(q_2, a)$ denotes the set of quadruples (q, w, x, t) where (q, w, x) is a triple in q_2 such that q admits a transition with input label a. $\nu(q_2, a)$ is the set of states q' that can be reached by transitions labeled with a from the states of the subset q_2. Notice that the state q could be $\{-1\}$ or $\{-2\}$ (for example in lines 7-8). *-1* and *-2* stand for the state number whereas $\{-1\}$ and $\{-2\}$ stand for the state with the number *-1* and *-2*. We know that q is the state number of the old transducer, q can never be a negative value. In fact, these states with a negative state number are generated and used to handle the special cases met during the determinization process. For easy understanding of this algorithm, a detailed introduction follows.

1. Line 1 refers that the initial final state set of T_2 is empty.

2. Lines 2-3 indicate that the initial weight λ_2 of T_2 is the minimum of all the initial weights of T_1. The initial state i_2 is a subset made of triples (i, ε, x), where i is an initial state of T_1, ε is an empty string, and $x = \lambda_1(i) - \lambda_2$. Fortunately, each transducer used in our automatic speech recognition research has only one initial state. This makes an easy implementation of lines 2-3. The next step is to put this initial state i_2 into an empty queue Q. Here, Q is used to maintain the set of subsets q_2 not yet extended (or determinized). Each subset in Q corresponds to one state of Q_2 for the new transducer T_2. Initially, Q contains only the subset i_2 (line 4).

3. F_2 is the set of the final states of the sequential transducer T_2. q_2 represents a final state iff it contains at least one triple (q, ε, x), where q is a final state of T_1 or equals to $\{-1\}$ (see lines 8-9), ε refers to an empty residual string, x is a residual weight. This type of triple is named *final triple*. The final output weight associated to q_2 is then the minimum output weight of all the final triples in q_2 (line 10).

4. q equal to $\{-1\}$ refers to a final state without outgoing arcs. This state does not exist. It is designed and assumed to be one special state of the old transducer during the determinization.

Line 11 meets the **special case 1**. In this case subset q_2 contains triple (q, w, x) such that $w \neq \varepsilon$, $q \in F_1$ or q equals $\{-1\}$. This type of triple is named *sub-final triple*. When a *sub-final triple* appears in q_2, a final state with accepting output (w, x) has been reached. According to the basic requirements, if each final state of the original transducer has only an accepting weight, the resulting transducer's final states can not have accepting output composed by string and weight. Accepting output of any final state of the resulting transducer has to be a weight.

How to handle it? Firstly a new subset q_2' has to be constructed with the triples (q, w, x) such that $w \neq \varepsilon$, $q \in F_1$ or q equals $\{-1\}$ in q_2. This q_2' is considered as part of q_2. Next we construct the set of outgoing output symbols w' from the set of w in q_2' (lines 12-13). Line 14 to 19 are used to finish the expanding based on w'. Each output symbol s in w' corresponds to an outgoing arc from q_2, the input symbol of this arc is ε, the weight is the minimum weight of all sub-final triples with same s in q_2' (lines 14-15). The destination of this arc is a subset $\delta_2(q_2, \varepsilon)$ formed by triples in q_2' with the same s. $\delta_2(q_2, \varepsilon)$ contains triples that q equals $\{-1\}$. The residual string of each triple in $\delta_2(q_2, \varepsilon)$ is the string by removing s from the residual string w of the corresponding triple in q_2'. The weight of each triple in $\delta_2(q_2, \varepsilon)$ is the result of the sum of the corresponding triple's residual weight x and its accepting cost in q_2' (if q equals $\{-1\}$ its accepting cost is zero) minus the output weight of the newly constructed outgoing arc. If $\delta_2(q_2, \varepsilon)$ is a new subset, then

put it into queue Q (lines 18-19). Continue this loop until each symbol in w' has been checked.

Therefore, q equals $\{-1\}$ means that q is considered as a special final state in the original transducer without outgoing arcs and with a final weight zero. It is special because that state $\{-1\}$ does not exist in Q_1. State $\{-1\}$ is needed during the determinization when subset q_2 has at least one *sub-final triple*.

5. For each input symbol a such that there exists at least one state q of the subset q_2 admitting an outgoing transition labeled with a, one temperate outgoing transition leaving q_2 with the input symbol a is constructed (lines 20–21). The output $\sigma_2'(q_2, a)$ of this temperate transition is composed of two parts. One is a residual string which is the largest common prefix of the output strings of all the transitions with input symbol a that leave a state in the subset q_2, when concatenated at the end with the residual string associated to that state. The other part is a residual weight which is the minimum of the output weights of all the transitions with input symbol a that leave a state in the subset q_2, when combined with the residual weight associated to that state.

The temporary destination state $\delta_2'(q_2, a)$ of the transition leaving q_2 is a subset made of triples (q', w', x'), where q' is a state of T_1 that can be reached by a transition labeled with a, w' is the corresponding residual string, x' is the corresponding residual weight. It is possible that $\delta_2'(q_2, a)$ has triples with same q' but different residual strings. Each residual string w' is constructed by removing the output string $\sigma_2'(q_2, a)|_w$ from the head of a string which is a concatenation of two strings. One of these two strings is the residual string w of the corresponding triple (q, w, x) in subset q_2, where q' can be reached from q by the transition $\sigma_1(t)$ with input symbol a. Another string is just the output string $\sigma_1(t)|_w$. x is computed by taking the minimum of output weights of all the transitions with input symbol a that leave a state q in the subset q_2 and reach the same state q' with the same residual string w', when combined with the residual weight of q minus the output weight $\sigma_2'(q_2, a)|_x$.

6. If $\sigma_2'(q_2, a)|_w$ is a symbol or an empty string, this temporary transition $\sigma_2'(q_2, a)$ is a transition leaving q_2 with the input symbol a named $\sigma_2(q_2, a)$, and the destination state for this transition is $\delta_2'(q_2, a)$ named $\delta_2(q_2, a)$ (lines 30–31). If $\sigma_2(q_2, a)$ is a new state then put it into queue Q.

Otherwise, if $\sigma_2'(q_2, a)|_w$ is a string with a length larger than 1, this temporary transition $\sigma_2'(q_2, a)$ has to be handled as a special case because of the basic requirements. This special case is defined as **special case 2**.

Review that according to the basic requirements during the determinization if the output of an outgoing arc A of a new state is a string (at least two symbols), A has to be transferred to the *symbol:symbol/weight* format. In *WT_determinization* this special case is handled by the following:

(1) Assign the string part $\sigma_2'(q_2, a)|_w$ to w''(line 24).

(2) Make a transition from q_2 with input symbol a, output symbol is the first symbol of w'', and its output weight is $\sigma_2'(q_2, a)|_x$ (line 25). Then, remove the first symbol of w''.

(3) The destination state $\delta_2(q_2, a)$ in line 27 is union of a new triple and $\delta_2'(q_2, a)$. The new triple $(\{-2\}, w'', 0)$, where $\{-2\}$ is a special state of q' which doesn't present in Q_1, it means that the state $\delta_2(q_2, a)$ is an expanding state of the string $\sigma_2'(q_2, a)|_w$, w'' is the residual string, and the residual weight is zero. Here, temporary state $\delta_2'(q_2, a)$ refers to that the final expanding state of this string is $\delta_2'(q_2, a)$. If state $\delta_2(q_2, a)$ is a new state, then enqueue it into queue Q (lines 28-29).

7. If the special case 2 described in the last paragraph occurs. A subset $(\{-2\}, w'', 0) \cup \delta'_2(q_2, a)$ will be assigned to q_2. Line 34 meets this condition. In line 35 a transition is made from q_2 with input symbol ε, output symbol is the first symbol of w'', and its output weight is zero. Then, remove the first symbol of w''. If w'' is an empty string, assign the state $\delta_2'(q_2, a)$ to $\delta_2(q_2, a)$. Otherwise, construct a destination state $\delta_2(q_2, a)$ by

the union of a new triple (*-2* as q', w'' as residual string, zero as residual weight) and the state $\delta_2'(q_2, a)$. The next step is to check whether state $\delta_2(q_2, a)$ is a new state. If $\delta_2(q_2, a)$ is a new state then put it into queue Q.

8. In line 42 the first element of Q is removed, and the algorithm returns to line 5. The algorithm will continue until Q is an empty queue. Finally when this algorithm is terminated a sequential weighted transducer T_2 is returned with the same functions of the non-sequential transducer T_1.

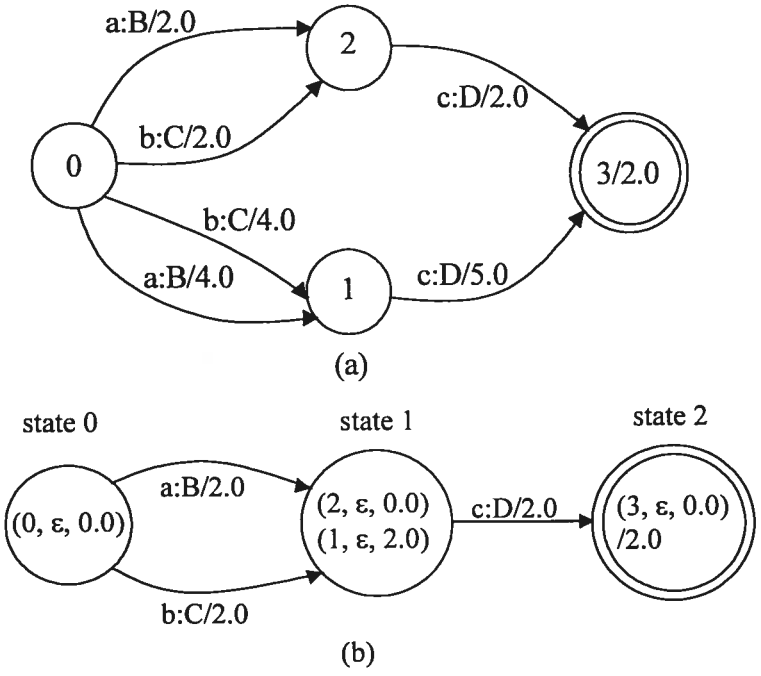
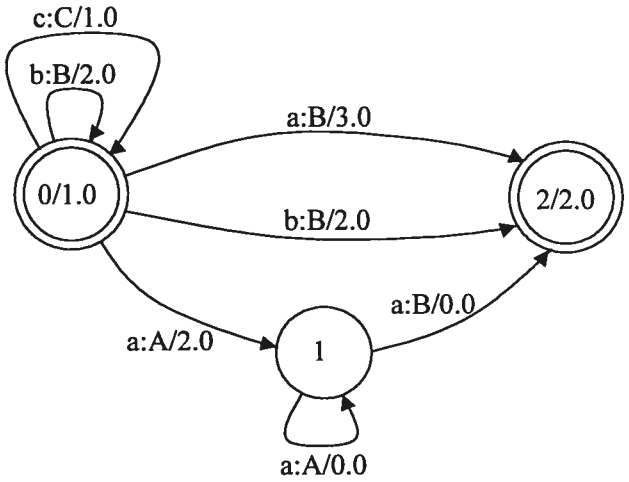
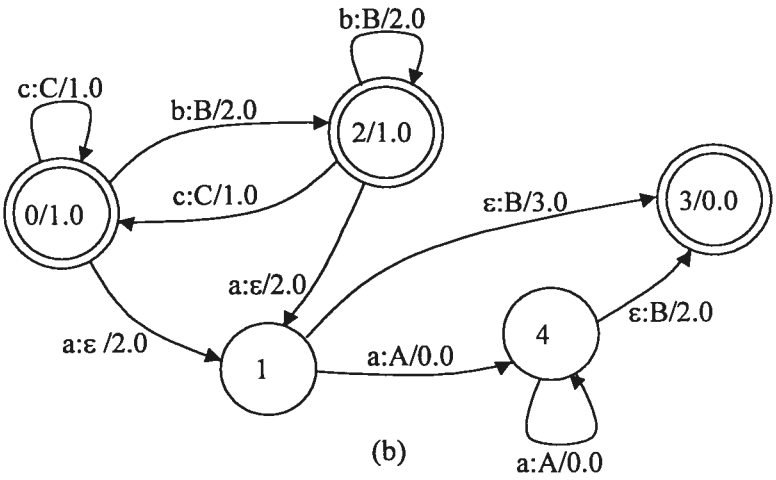


Figure 3.9 (a) a non-sequential weighted transducer T_1
(b) a sequential weighted transducer T_2 obtained from the *WT_determinization* of T_1

9. Examples of this algorithm are shown in Figure 3.9 and Figure 3.10. Notice that an input string *ac* admits several outputs in T_1 in Figure 3.9: {(BD, 6), (BD, 11)}. Only one of these outputs ((BD, 6), with the smallest output weight) is kept in the resulting sequential transducer T_2 since we are only interested in the output with the minimum output weight for any given string.



(a)



(b)

Figure 3.10 (a) a non sequential weighted transducer T_1.

(b) sequential transducer T_2 obtained from the determinization of T_1

In Figure 3.9, after the determinization both state number and arc number are reduced. However, in some special case the state number or both state number and arc number is/are increased after the determinization such as in Figure 3.10. This is also a successful

determinization result because the resulting transducer is sequential. The sequential transducer will dramatically increase the searching speed when it is applied to the CSR process instead of the equivalent non-sequential one [3].

Notice that several transitions might reach the same state with a same residual string but a priori different residual weights. Since only the best path is interested, namely the path corresponding to the minimum weight, the algorithm keeps only the minimum of these weights for a given state element of a subset (line 22). This case can be observed in the final determinization step of T_1 in Figure 3.9.

3.3.2.2 A Proof of the WT_determinization Algorithm

It is very important to prove that if the determinization algorithm terminates then the resulting sequential transducer T_2 is equivalent to T_1.

Here, we emphasis the termination of the determinization algorithm. This is because there are transducers for which determinization does not halt. It then generates an infinite number of subsets. We define determinizable transducers as those transducers with which the algorithm terminates.

Assume that the determinization algorithm terminates, then the resulting transducer T_2 is equivalent to T_1. The following is the proof [18].

We denote by $\theta_1(q, w, q', w')$ the output with the minimum weight of all paths from q to q' with a same output string w'. By construction we have:

$$\lambda_2 \leftarrow \min_{i \in I_1} \lambda_1(i)$$

We define the residual output string $s(q, w)$ and the residual output weight associated to q in the subset $\delta_2(i_2, w)$ as the weight $c(q, w)$ associated to the triple containing q in $\delta_2(i_2, w)$. By induction on $|w|$ we show that the subsets constructed by the algorithm are the sets $\delta_2(i_2, w)$, $w \in \Sigma^*$, such that:

$$\forall w \in \Sigma^*,\ \delta_2(i_2, w) = \bigcup_{q \in \delta_1(I_1,\ w)} \{(q, s(q, w), c(q, w)\} \qquad (3.1)$$

$$c(q, w) = \min_{i_1 \in I_1} (\lambda_1(i_1) + \theta_1(i_1, w, q, w')) - \sigma_2(i_2, w)|_x - \lambda_2$$

$$\sigma_2(i_2, w) = (w' \bullet s(q, w)^{-1},\ \min_{q \in \delta_1(I_1,\ w)} (\lambda_1(i_1) + \theta_1(i_1, w, q, w',)) - \lambda_2)$$

A pair $(q, s(q, w))$ belongs at most to one triple of a subset since for all paths reaching q with a same residual string $s(q, w)$, only the minimum of the residual output weight is kept. If $s(q, w)$ equals to an empty string, the output string of $\sigma_2(i_2, w)$ is w'. Notice also that, by definition of min, in any subset there exists at least one state q with a residual output weight $c(q, w)$ equal to 0.

A string w is accepted by T_1 iff there exists $q \in F_1$ such that $q \in \delta_1(I_1, w)$. Its output string is w'. Using the equation (3.1), it is accepted iff $\delta_2(I_2, w)$ contains a triple $(q, s(q, w), c(q, w))$ with $q \in F_1$ and $s(q, w)$ equals to an empty string. This is exactly the definition of the final states F_2 (line 8). So T_1 and T_2 accept the same set of strings. Therefore, T_2 and T_1 are equivalent.

3.3.2.3 Space and Complexity of the WT_determinization Algorithm

Both space and time complexity of the determinization algorithm for the determinizable weighted transducers are exponential to the size of the original transducer T_1. However, in some cases in which the degree of non-determinism of the initial transducer is high, the determinization algorithm turns out to be fast because the resulting transducer has much less states than the initial one.

The proof on the space and time complexity of the *WT_determinization* is the same as that of the determinization of automata or weighted acceptors. It can be found in references [18, 22]. Formerly, the definitions of the weighted transducers have been addressed. And the *WT_determinization* algorithm for the determinization of the determinizable weighted transducers has been provided.

Since the *WT_determinization* algorithm returns the whole resulting transducer, a large amount of memory is needed for the determinization, especially when the size of the resulting transducer is large.

3.4 Minimization

Any deterministic acceptor can be minimized using classical minimization algorithms [2, 22]. Similarly using the minimization algorithm for WFSAs, we can minimize any deterministic weighted acceptor A [20, 22]. After applying minimization algorithm on WFSA A, the resulting weighted acceptor B has the least number of states and the least number of transitions among all deterministic weighted acceptors equivalent to A.

A deterministic weighted acceptor A can be viewed as an unweighted acceptor by interpreting each pair (a, w) of a label a and a weight w as a single label. We can then apply the classical minimization algorithm to this acceptor. But, it could have no effect on A since the pairs for different transitions are all distinct. In the same way we can also minimize deterministic transducers or weighted transducers.

However, before applying classical minimization algorithms, we can transform A to an equivalent weighted acceptor A_1 by using a pushing algorithm which pushes weight among arcs. Usually the weights are pushed towards the initial state as much as possible. The size of A_1 can then still be reduced by applying the classical minimization algorithm to the resulted transducer A_2 with each distinct label-weight pair viewed as a distinct symbol, as described above. The details about the weight pushing algorithm will be discussed in Chapter 5.

Figure 3.11 shows a example of a weighted acceptor A, Figure 3.12 shows a weighted acceptor A_1 obtained by *pushing* from A in the tropical semiring. And Figure 3.13 shows the weighted acceptor A_2 obtained by *minimizing* A_1. Obviously the resulting automaton A_2 is equivalent to A_1 and the size of the automaton can then be reduced using the minimization algorithm.

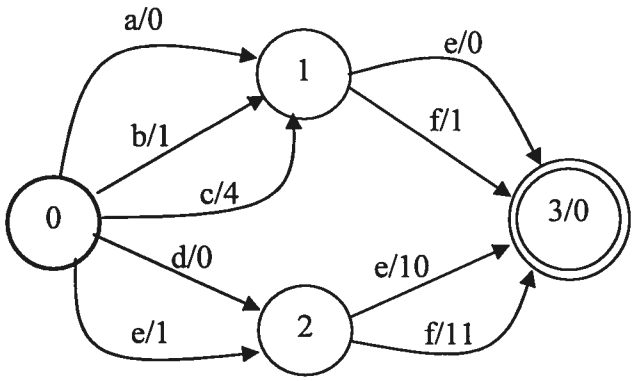


Figure 3.11: Weighted acceptor A

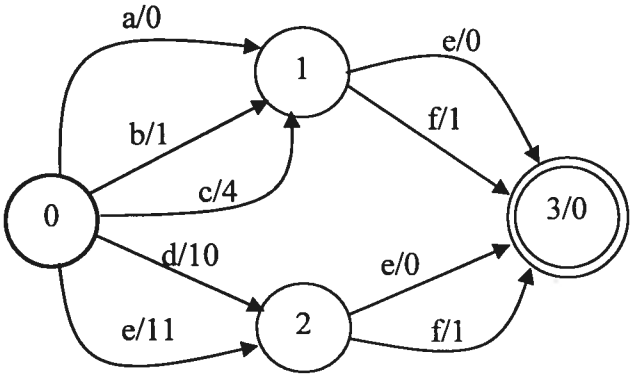


Figure 3.12: Weighted acceptor A_1 obtained by *pushing* from A in the tropical semiring

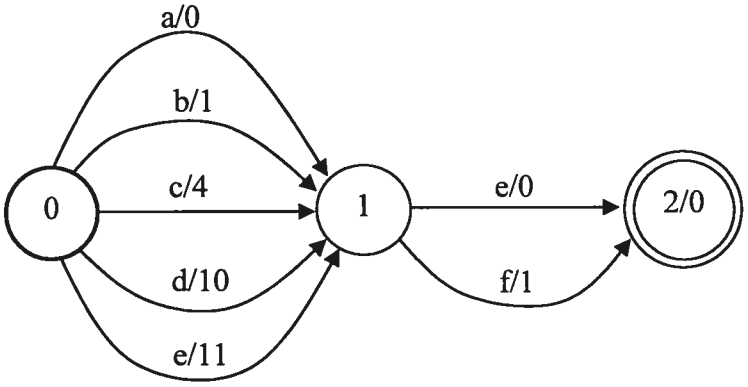


Figure 3.13: Weighted acceptor A_2 obtained by *minimizing* A_1

The time complexity of weighted minimization is the same as that of classical minimization: linear in the acyclic case ($O(m+n)$) [5], and ($O(m \log n)$) [2] in the general case, where n is the number of states and m the number of transitions.

In this chapter we have introduced the transducer operations. How can they be used in speech recognition? In the next chapter we will describe the application of weighted transducers in speech recognition.

Chapter 4

Weighted Finite-State Transducer Applications in Speech Recognition

In Chapter 3 we have described some weighted finite-state transducer operations. These operations are very useful in speech recognition when all of the components of the recognition network are represented by transducers. In this chapter we will describe the applications of weighted finite-state transducers in speech recognition.

4.1. Network Components

The speech recognition networks can be combined and optimized as an integrated speech recognition network prior to decoding. The word pronunciations are found in the lexicon and are substituted into the grammar. The decoder of the recognition networks then identifies the correct context-dependent models to use for each phone in context, and finally substitutes them to create an HMM-level network. The code that performs these operations is specified to particular model topologies. For example, the context-dependent models might have to be triphonic, the grammar might be restricted to trigrams, and the alternative pronunciations might have to be enumerated in the lexicon. Moreover, these network combinations and optimizations are applied in an appropriate sequence to a prespecified number of levels as we will describe later in this chapter.
However a uniform transducer representation can be used for n-gram grammars, pronunciation dictionaries, context-dependency specifications, HMM topology, word, phone or HMM segmentations, lattices and n–best output lists. In the following sections, the transducer representations in speech recognition will be discussed in detail.

4.1.1. Transducer O

The acoustic observation can be represented as a weighted acceptor O for a given utterance as shown on Figure 4.1. Each state represents a fixed point in time t_i, and each

transition has a label, o_i, drawn from a finite alphabet that quantifies the acoustic signal between adjacent time points and is assigned probability 1.0.

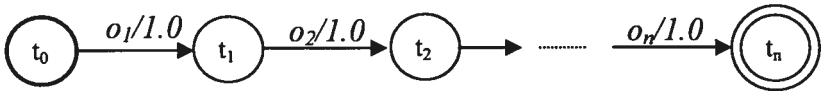


Figure 4.1 Weighted acceptor for acoustic observation

The weighted acceptor in Figure 4.1 can be interpreted as a weighted transducer by giving each transition identical input and output labels. This adds no new information, but is a convenient way of interpreting any acceptor as a transducer.

For more complex acoustic distributions (for instance, continuous densities) we can instead use multiple transitions (t_{i-1}, d, $p(o_i|d)$, t_i) where d is an observation distribution and $p(o_i|d)$ the corresponding observation probability.

4.1.2. Transducer H

Figure 4.2 shows a three-state HMM transducer as a phone model mapping sequences of distribution indices to context-dependent phones, i.e. triphones. In this figure, the triphone is denoted as a-b+c where b is the modeled phoneme, a and c are the left and right neigboring phonemes of b, and O_i denotes the observation distribution, d_{ij} denotes transition probabilities. The transducer H is obtained by taking all the closure of the union of all HMMs used in acoustic modeling. Therefore transducer H preserves phone model identity while sharing distribution subsequences whenever possible.

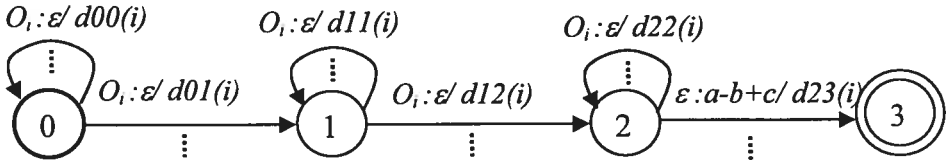


Figure 4.2 A HMM transducer for a context-dependent model phone b

4.1.3. Transducer C

Context-dependent phone models can be represented by finite-state transducers. They can be built directly in many cases. The transducer of Figure 4.3 for instance can be constructed directly to represent the inverse of a context-dependency model: it maps phone sequences to sequences of names of context-dependent phone models (HMMs). It encodes triponic context-dependency for only two hypothetical phones x and y for simplicity. Each state (a, b) encodes the information that the previous phone was a and the next phone is b; ε represents the start or end of a phone sequence and * an unspecified next phone.

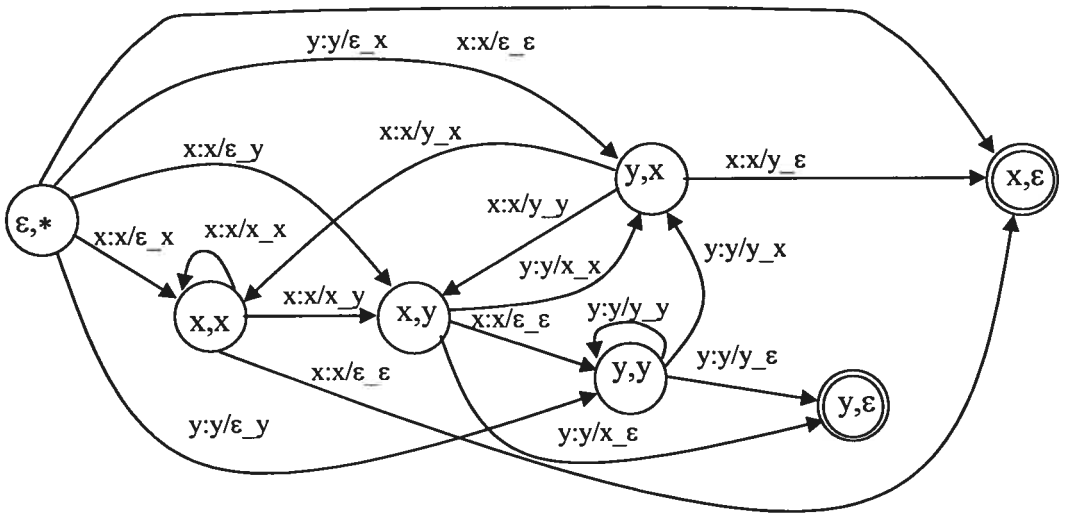


Figure 4.3: Context-dependent triphone transducer

For instance, it is easy to see that the phone sequence xyx is mapped by the transducer to $x/\varepsilon\_y\ y/x\_x\ x/y\_\varepsilon$ via the unique state sequence $(\varepsilon, *)(x, y)(y, x)(x, \varepsilon)$. More generally, when there are n context-independent phones, this triphonic construction gives a transducer with $O(n^2)$ states and $O(n^3)$ transitions. A tetraphonic construction would give a transducer with $O(n^3)$ states and $O(n^4)$ transitions. In real applications, context-dependency transducers will benefit significantly from determinization and minimization since the n-phone is modeled by an HMM that is likely to be shared among many n-phones due to context clustering required by data sparsity.

4.1.4. Transducer L

The transducer L from phone sequences to word sequences is built similarly to transducer H. A word model is a transducer from phone sequences to the specified word that assigns to each phone sequence the likelihood that the specified word produced. Thus, different paths through a word model correspond to different phonetic realizations of the word. Figure 4.4 shows a typical topology for a word model. L is then defined as the closure of the sum of the word models. Figure 4.5 shows a toy pronunciation lexicon.

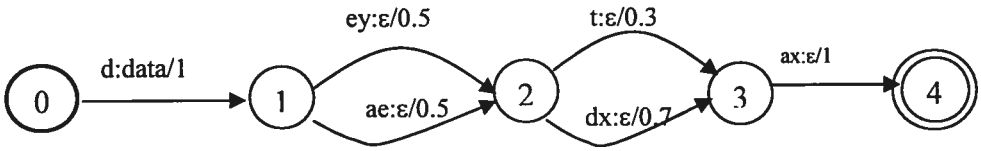


Figure 4.4 A word model as weighted transducer

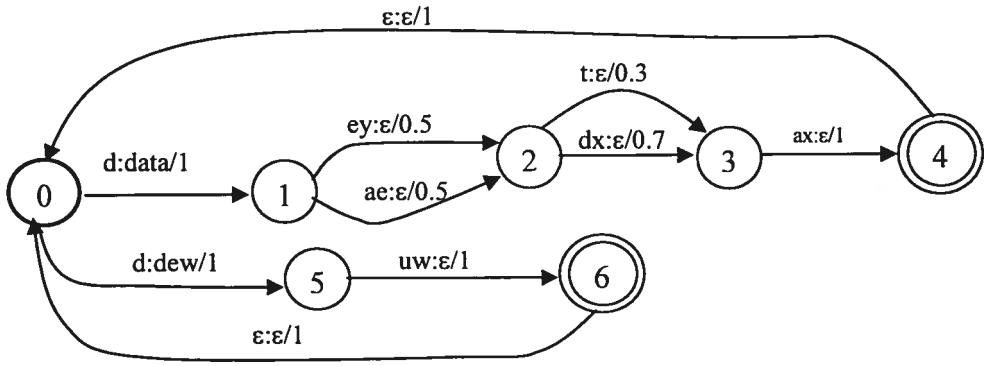


Figure 4.5 A toy pronunciation lexicon of transducer L

4.1.5. Transducer G

The language model can be easily represented as a weighted acceptor as shown in Figure 2.4. The following weighted transducer in Figure 4.6 is an interpretation of the weighted acceptor in Figure 2.4 as a weighted transducer. Any n-gram language model can be represented in a similar way. An n-gram language model G is often constructed as a deterministic weighted acceptor with a back-off state — in this context, the symbol ε is treated as a regular symbol for the definition of determinism.

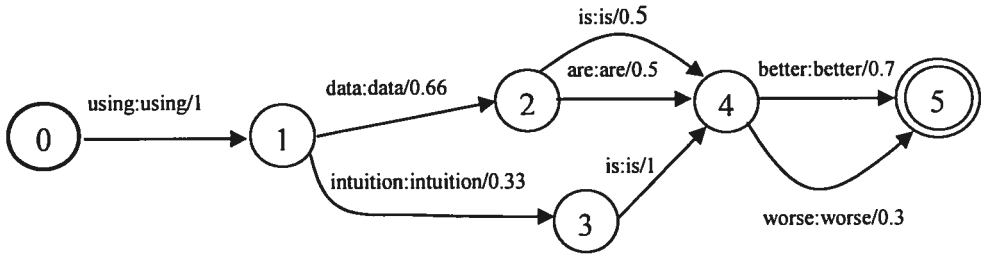


Figure 4.6 A toy language model as a weighted transducer

4.2. Network Combination

The network components can be combined by applying composition operations. Consider the word model in Figure 4.4. The transducer L is formed by taking the union of this transducer with the transducers for the remaining word models in the grammar G of Figure 4.6, and then taking its Kleene closure by connecting an ε-transition from each final state to the initial state. The resulting transducer L would pair any sequence of words from that vocabulary to their corresponding pronunciations. Thus,

$$L \circ G$$

gives a transducer that maps from phones to word sequences restricted to G.

The composition can also be used to implement a context-independent substitution. However, a major advantage of transducers for speech recognition is that they generalize naturally the notion of context-independent substitution of labels by a network to the context-dependent case. The transducer of Figure 4.3 does not correspond to a simple substitution, since it describes the mapping from context-independent phones to context-dependent triphonic models, denoted by *phone/left context_right context.*

The following simple example shows the use of this context-dependency transducer. A context-independent string can be represented by the obvious automaton having a single path as in the example in Figure 4.7a. We thus can interpret it as a corresponding transducer with identical input and output labels. Then we compose it with the context-dependency transducer in Figure 4.3. The result is the transducer in Figure 4.7b, which has a single path labeled with the context-independent labels on the input side and the corresponding context-dependent labels on the output side.

The context-dependency transducer, of course, can be composed with more complex networks than the trivial one in Figure 4.7a. For example, composing the context-dependency transducer with the transducer in Figure 4.7c results in the transducer in Figure 4.7d. By definition of relational composition, this must correctly replace the context-independent units with the appropriate context-dependent units on all its paths. Therefore, composition provides a convenient and general mechanism for applying context-dependency to CSR networks.

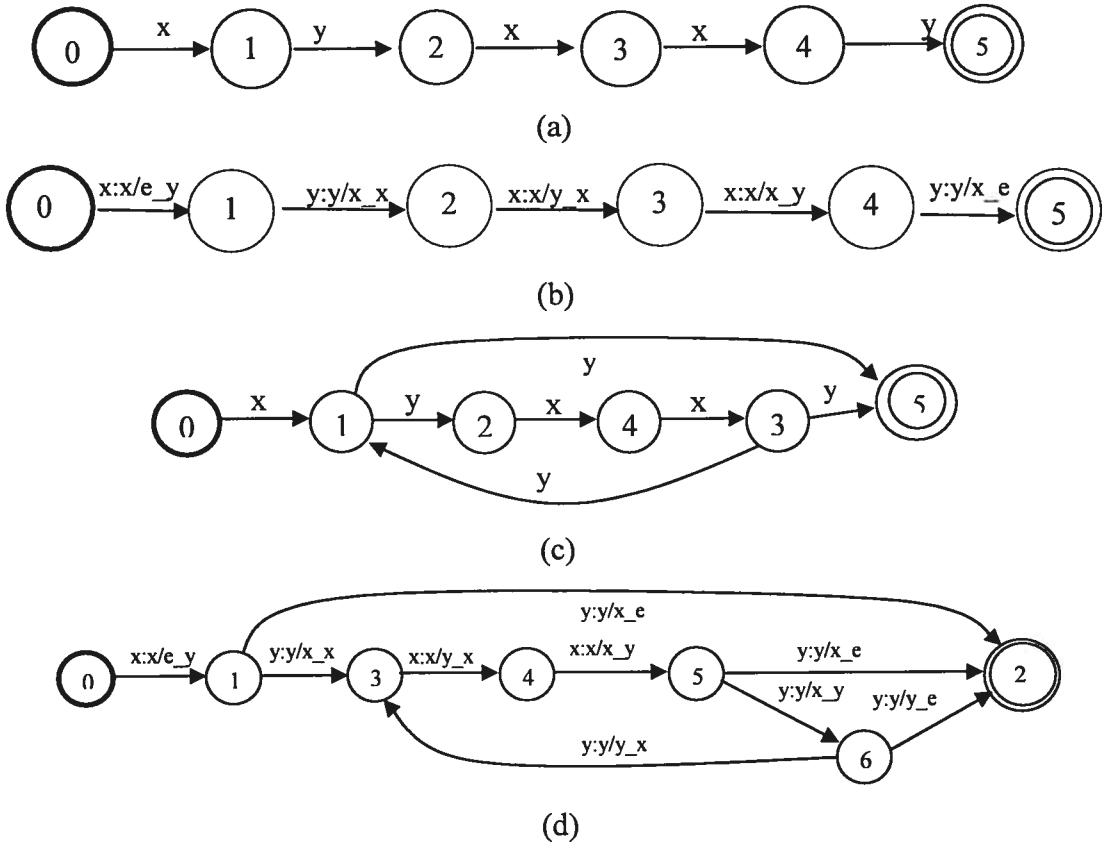


Figure 4.7: Context-dependent composition examples

Let C represent a context-dependency transducer from context-dependent phones to context-independent phones, then

$$C \circ L \circ G$$

gives a transducer that maps from context-dependent phones to word sequences restricted to the grammar G. Note that C is the inverse of a transducer such as in Figure 4.3.

The context-dependency transducer, of course, can be composed with more complex networks than the trivial one in Figure 4.7a. For example, composing the context-dependency transducer with the transducer in Figure 4.7c results in the transducer in Figure 4.7d. By definition of relational composition, this must correctly replace the context-independent units with the appropriate context-dependent units on all its paths. Therefore, composition provides a convenient and general mechanism for applying context-dependency to CSR networks.

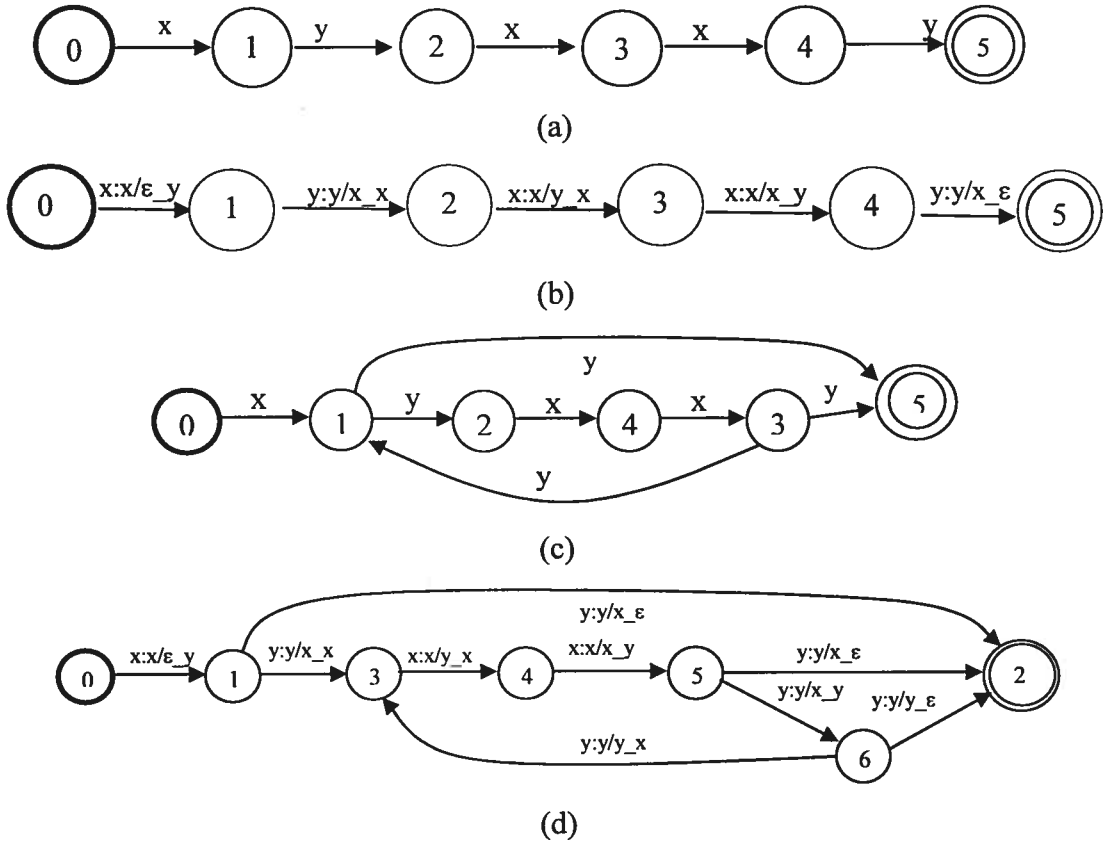


Figure 4.7: Context-dependent composition examples

Let C represent a context-dependency transducer from context-dependent phones to context-independent phones, then

$$C \circ L \circ G$$

gives a transducer that maps from context-dependent phones to word sequences restricted to the grammar G. Note that C is the inverse of a transducer such as in Figure 4.3.

As the pronunciation lexicon, the HMM set is represented as H, the closure of the union of the individual HMMs as in Figure 4.2. With H in hand,

$$H \circ C \circ L \circ G$$

gives a transducer that maps from distributions to word sequences restricted to G.

Thus composition can be used to combine all levels of the CSR network into an integrated network in a convenient, efficient, and general manner. Then the optimizations are applied to reduce decoding time and space requirements. If the network needs to be modified dynamically, for example by adding the results of a database lookup to the lexicon and grammar in an extended dialogue, a hybrid approach is adopted so that it can optimize the fixed parts of the network and can use lazy composition to combine them with the dynamic portions during recognition.

4.3. Network Optimization

The benefits of transducer application in speech recognition are time and space efficiency, which rely on the transducer optimization operations. Here two steps are described in detail for optimizing an integrated network; (a) determinization, and (b) minimization.

4.3.1. Determinization

The weighted transducer determinization is used at each step of the composition of each pair of networks. The determinization eliminates redundant paths in the composed network to reduce recognition time. Furthermore, its use in intermediate steps of the construction also helps to improve the efficiency of composition and to reduce network size.

It can be shown that, in general, the transducer $L \circ G$ from phone sequences to words is not determinizable. This is clear in presence of *homophones*. Homophones are words of the same language that are pronounced alike even if they differ in spelling, meaning, or origin, such as "pair" and "pear". Homophones may also be spelled alike, as in "bear" (the animal) and "bear" (to carry). But, even in the absence of homophones, $L \circ G$ might not be determinizable because, in some cases, the first word of the output sequence cannot be

determined before the entire input phone sequence is known. Such unbounded output delays make $L \circ G$ non-determinizable.

An auxiliary phone symbol, denoted $\#_0$ which marks the end of the phonemic transcription of each word, is introduced to make it possible to determinize. Other auxiliary symbols $\#_0 \ldots \#_{k-1}$ are used when necessary to distinguish homophones as in the following example:

r eh d $\#_0$ *read*

r eh d $\#_1$ *red*

At most P auxiliary phones, where P is the maximum degree of homophony, are introduced. The pronunciation dictionary transducer augmented with these auxiliary symbols is denoted by $\tilde{L}$.

For consistency, the context-dependency transducer 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 C, mapping each auxiliary phone to a new auxiliary context-dependent phone. The augmented context-dependency transducer is denoted by $\tilde{C}$.

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

The addition of auxiliary symbols guarantees the determinizability of the transducer obtained after each composition, allowing the application of weighted transducer determinization at several stages in the construction.

First, $\tilde{L}$ is composed with G and determinized, yielding $det(\tilde{L} \circ G)$. 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, while the original network may have as many as V outgoing transitions at some states where V is the vocabulary size. For large

tasks in which the vocabulary has 10^5 to 10^6 words, the advantages of the optimization are clear [29].

The context-dependency transducer might not be deterministic with respect to the context-independent phone labels. For example, the transducer shown in Figure 9 is not deterministic since the initial state admits several outgoing transitions with the same input labels x or y. To build a small and efficient integrated network, it is important to first determinize the inverse of $\tilde{C}$.

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

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

$$N = \pi_\varepsilon(det(\tilde{H} \circ det(\tilde{C} \circ det(\tilde{L} \circ G)))) \quad (4.1)$$

where parentheses indicate the order in which the operations are performed. The result N is an integrated recognition network that can be constructed even in very large-vocabulary tasks, and leads to a substantial reduction of the recognition time [29].

4.3.2. Minimization

After being determinized, the integrated network can be further reduced in size by being minimized. We keep the auxiliary symbols in place, and then apply the minimization algorithm. After that, we can remove the auxiliary symbols:

$$N = \pi_\varepsilon(min(det(\tilde{H} \circ det(\tilde{C} \circ det(\tilde{L} \circ G))))) \quad (4.2)$$

Minimization only reduces the size of the transducers in terms of number of states and transitions, but also significantly affects the decoding speed. As described earlier, a key

step of minimization is to push weights toward the initial state. The pruning, which uses a conventional Viterbi beam search for the decoder, is also largely affected by this. It slows down decoding if pushing in the tropical semiring in some cases [28], but in our tests it gives good results. However, it surely has a beneficial effect on the decoding speed if pushing in the log semiring, which is that the negative log of the sum of all probability mass from each state to the (super-) final state is used as the reweighting potential function. When this potential function is used for reweighting, the total sum of probabilities over all transitions leaving any state is 1. Thus, the transducer is pushed in terms of probabilities along all future paths from a given state rather than the highest probability over the single best path. Interestingly, it can be proved that using either pushing in the minimization step results in equivalent machines. However, by using log probability pushing (pushing in the log semiring), a property that holds for the language model now also holds for the integrated network, namely the weights of the transitions leaving each state are normalized as in a probabilistic automaton.

If the lowest cost path potential function is used, then classical single-source shortest path algorithms can be employed, which we will describe in Chapter 5. However, adopting the sum of probability mass potential function required a significant generalization, of independent interest, to the classical algorithms [28].

In the WFSTs approach for CSR, since the search transducer is built statically, these optimizations can be performed entirely off-line. The transducers in recognition network are handled in a highly flexible way, independently of the decoder specifics about contextual range. The final optimized network is substantially reduced to only twice that of language model [27]. Cross-word context expansion increases the network by just a few percents with respect to the optimized context-independent network [27].

In this chapter we have introduced the application of weighted transducers in speech recognition. The transducers in the recognition network can be combined and then standardized in order to increase time and space efficiency. In the next chapter we will present a transducer compaction operation which can reduce the size of non-determinizable transducers.

Chapter 5

The Compaction of Finite-State Transducers

In the last chapter we have surveyed weighted finite-state transducers in speech recognition. Weighted determinization and minimization can optimize time and space requirements of a transducer. However as aforementioned, not all transducers are determinizable, even the resulted transducer N of eq.4.2. In cases when a transducer or a weighted acceptor can not be determinized, the transducer determinization and minimization algorithms described in Chapter 3 cannot be applied. It is necessary to find an operation to reduce the size of the transducer in order to further increase time and space efficiency in speech recognition. We developed a *finite-state transducer compaction algorithm* which can apply on non-determinizable transducers to possibly reduce the size of the transducers. It is very useful in speech recognition as we will discuss later.

5.1 Transducer Compaction

Any automaton (unweighted acceptor) can be determinized by applying classical automata determinization operations and then can be minimized by applying classical automata minimization [2, 30]. The transducer determinization and minimization can only apply on determinizable transducers. To reduce the size of a non-determinizable transducer, the considered transducer can conceptually be converted into an automaton. Theoretically if we consider all the symbols and cost of an arc as a single label, the transducer can be considered as an automaton. Therefore by applying classical automata determinization and minimization on the "converted" transducers, i.e. the new automata, the original transducers may be possibly reduced in size. However a weight pushing operation can also be used before converting the transducer into automaton, and then the classical automata determinization and minimization can be used to reduce its size as much as possible. This operation is called *transducer compaction*. The transducer compaction operation has the following six steps:

1. Weight pushing: to push the weight towards the initial state as much as possible for weighted transducers or acceptors.
2. Encoding: to convert a transducer into an automaton, that is to encode each triple of an input label, output label, and cost (weighted transducers) or each double of an input and cost or output (weighted acceptors or string-to-string transducers) into a single new label.
3. Determinization checking: to check the determinism of the new automaton. If it is deterministic skip the determinization step, i.e. step 4, and go to minimization directly.
4. Determinization: to apply the classical automata determinization on the new automaton..
5. Minimization: to apply classical automata minimization on the deterministic automata.
6. Decoding: the encoded labels are decoded back into their original values.

5.2 The Automata Determinization

The well-known powerset construction from [30] shows that for any non-deterministic finite automaton (NFA) there is an equivalent deterministic finite automaton (DFA). The main idea of the construction is to consider sets of states of the NFA as states of the DFA. The DFA uses these sets to remember the set of states that the NFA could have reached after reading the same input.

5.2.1 The Automata Determinization Algorithm

Let the input finite automaton $A_1=(\Sigma_1, Q_1, I_1, F_1, E_1)$ and the output deterministic finite automaton $A=(\Sigma, Q, \{i\}, F, E)$. A is the result of the *determinization* of A_1. $det(A_1)$, accepting $L(A_1)$.

The pseudocode of the automata determinization is as follows:

1. $F \leftarrow E \leftarrow 0$
2. $i \leftarrow I_1$
3. $S \leftarrow Q \leftarrow \{I\}$
4. *while* $S \neq \emptyset$
5. *do* $s \leftarrow$ *head[S]*

```
6.          if s∩F_I ≠ Ø
7.              then F← F ∪{s}
8.          for each a ∈ Σ such that δ_I(s, a) ≠ Ø
9.            do if δ_I(s, a) ∉ Q
10.               then Q← Q ∪{δ_I(s, a)}
11.                   ENQUEUE (S, δ_I(s, a))
12.           E←E ∪ {(s,a, δ_I(s, a))}
13.     DEQUEUE (S)
```

Figure 5.1 The automata determinization algorithm

5.2.2 The Complexity of the Automata Determinization Algorithm

1. For loop of lines 4-13: each subset s is enqueued at most once (test line 9), hence the loop is executed at most as many times as the number of states created $|Q|$ ($|Q|<2^{|Q_I|}$).
2. Test line 6: proportional to the size of s: $O(|s|)$,
3. Loop 8-13: the number of iterations can be made independent of Σ. By considering $E_1[s] = \cup_{q \in s} E_1[q]$,
   (a) the loop is executed at most $|E|$ times,
   (b) line 8:
      i. in general, one needs to sort the transitions to determine the set of transitions of $E_1[s]$ labeled with each a: $O(|E_1[s]|\log(|E_1[s]|))$,
      ii. if the transitions of A_1 are presorted with respect to a total order defined on Σ, a k-merge of $E[q]$, $q \in s$, can be used: $O(|E_1[s]|\log(|s|))$,
   (c) line 9:
      i. the test can be done by sorting the elements of $\delta_1(s,a)$: $O(|\delta_1(s,a)|\log(|\delta_1(s,a)|))$, and then using a direct-address table or using a perfect hashing: $O(|\delta_1(s,a)|)$,
      ii. if the transitions are presorted with respect to Σ and the destination states, and if a k-merge is used, no additional sorting is necessary,
      iii. in practice, a hash table is used for the membership test and for assigning numbers to the states,

4. Total complexity:

(a) $O(Q\max_{s\in Q}|s| + \Sigma_{s\in Q}(|E_1[s]|\log|s|) + |E|\max_{s\in Q}|s|)$

$= O(|Q_1|(|Q| + |E| + |E_1|\log|Q_1|)\}$

(b) Exponential with respect to the size of A_1: $Q = O(2^{|Q_1|})$.

It is easy to see that the first term in (a) dominates the running time of the algorithm, hence the total complexity is: $O(|Q_1|\cdot 2^{|Q_1|})$.

5.3 Weight Pushing

Weight Pushing consists in pushing all the weights along the path towards the initial state as much as possible. It is a special case of *reweighting*.

5.3.1 Reweighting

Here we will describe the reweighting in the case of the tropical semiring, a similar definition can be given in the case of other semirings. There are infinite ways of reweighting for a weighted automaton to produce equivalent automata. Given a weighted acceptor $T = (Q, \Sigma, i, F, E, \lambda, \rho)$ over the semiring K as defined in section 2.2.7.1 we can let a transition $t = (p[t], l[t], w[t], n[t]) \in E$ represent an arc from the source state $p[t]$ to the destination state $n[t]$, with the label $l[t]$ and weight $w[t]$. In speech recognition, the transition weight $w[t]$ often represents a probability or a log probability.

We can change the transition weights of a weighted acceptor T without modifying path weights [27]. To see how it works, let $V: Q \to K - \{\bar{0}\}$ be an arbitrary function, called a potential function on states. Update the initial weight, the transition weights, and the final weights by:

$$\lambda \leftarrow \lambda \otimes V(i) \tag{3.1}$$

$$\forall e \in E,\ w[e] \leftarrow [V(p[e])]^{-1} \otimes (w[e] \otimes V(n[e])) \tag{3.2}$$

$$\forall f \in F,\ \rho(f) \leftarrow [V(f)]^{-1} \otimes \rho(f) \tag{3.3}$$

It is easy to see that this reweighting does not affect the total weight of a successful path and that the resulting automaton is equivalent to the original since the potentials along any successful path cancel each other.

Weight pushing consists of reweighting an automaton with the potential V defined in such a way that for each state $q \in Q$, $V(q) = d[q]$, the *shortest distance* from q to the final states F defined by:

$$d[q] = \oplus_{\pi \in P(q)} w[\pi]$$

where $P(q)$ represents the set of all paths from q to F. For the case of semirings, we can assume that the semiring K is *divisible* [28], that is for any $a, b \in K$ such that $a \oplus b \neq \bar{0}$ there exists $a_l \in K$ such that:

$$a = (a \oplus b) \otimes a_l$$

When K is divisible, for each pair (a, b) such that $a \oplus b \neq \bar{0}$, a fixed element a_l can be selected to satisfy this equation. We call a_l *the remainder of the division of a by* $a \oplus b$ and write:

$$a_l = (a \oplus b)^{-1} \otimes a$$

Figure 5.3 shows the result of pushing for the input automaton of Figure 5.2 defined over the tropical semiring. As can be seen from Figure 5.3, the shortest path from each state to a final state has weight zero. This is a general consequence of pushing in the tropical semiring. Figure 5.4 shows the result of pushing in the log semiring applied to A_l. At each state, the probability weights of outgoing transitions sum to one.

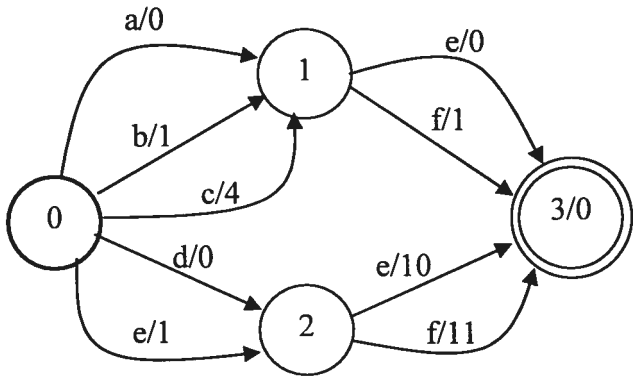


Figure 5.2: Weighted acceptor A_l

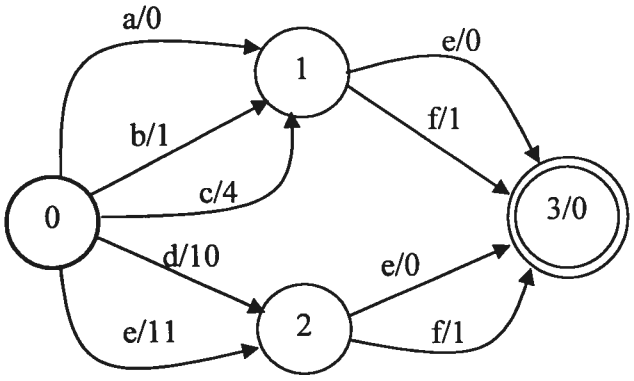


Figure 5.3: Weighted acceptor A_2 obtained by *pushing* from A_1 in the tropical semiring

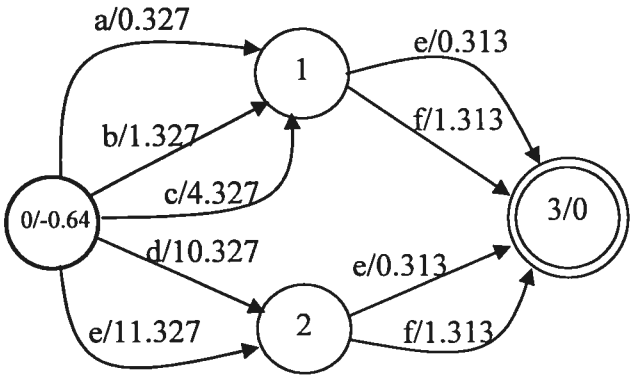


Figure 5.4: Weighted acceptor A_3 obtained by *pushing* from A_1 in the log semiring

The key step in weight pushing is how to compute the potential function. In the case of the tropical semiring, the shortest distance from each state to final states can be computed efficiently using classical single-source shortest-paths algorithms [23].

In the case of probability semiring or the log semiring, a modified version of the Floyd-Warshall algorithm can effectively compute shortest distances under certain general conditions [23]. But, this algorithm has time complexity $O(|Q|^2)$ and space complexity $O(|Q|^3)$. This can make its application to large transducers in speech recognition impossible in practice. However, a generic single-source shortest-distance (SSSD) algorithm can work with any *k-closed* semiring [23]. A *k*-closed semiring is a semiring for which there exists *k* such that:

$$\forall a \in K, \bigoplus_{n=0}^{k+1} a^n = \bigoplus_{n=0}^{k} a^n$$

We can see that the tropical semiring is *0*-closed. The log semiring is not k-closed for any $k \in N$. However, if the equality tests in the SSSD algorithm are replaced by approximate equalities modulo a small number ε, then the SSSD algorithm can also cover the case of the log semiring by producing ε-approximations of the correct shortest distances. Smaller ε's result in better approximations and in practice, the resulting approximation is in the order of or less than that of the approximations already made in speech recognition [28]. In the case of the log semiring, the SSSD algorithm is several orders of magnitude faster than the extension of the Floyd-Warshall algorithm and is very practical even for very large recognition transducers. In the case of tropical semiring, the SSSD algorithm coincides with classical shortest-distance algorithms.

5.3.2 Weight Pushing Pseudocode

GENERIC-SINGLE-SOURCE-SHORTEST-DISTANCE (WEIGHTED ACCEPTOR T)

```
1  for j←1 to |Q|
2       do d[j] ← r[j] ← 0̄
3  d[i] ← r[i] ← 1̄
4  S← {i}
5  while S≠0
6       do q← head(S)
7          DEQUEUE (S)
8          R← r[q]
9          r[q] ← 0̄
10         for each e ∈ E[q]
11             do if d[n[e]]≠d[n[e]] ⊕ (R ⊗ w[e])
12                    then d[n[e]] ← d[n[e]] ⊕ (R ⊗ w[e])
13                         r[n[e]] ← r[n[e]] ⊕ (R ⊗ w[e])
14                         if n[e] ∉ S
15                            then ENQUEUE(S, n[e])
16 d[i] ← 1̄
```

Figure 5.5: Generic single-source shortest-distance algorithm

Figure 5.5 gives the pseudocode of the algorithm. A queue S is used to maintain the set of vertices whose leaving edges are to be *relaxed*. S is initialized to $\{i\}$ (line 4). For each vertex $q \in Q$, two attributes are maintained: $d[q] \in K$ an estimate of the shortest distance from i to q, and $r[q] \in K$ the total weight added to $d[q]$ since the last time q was extracted from S. Lines 1-3 initialize arrays d and r. After initialization, $d[q] = r[q] = \bar{0}$ for $q \in Q - \{i\}$, and $d[q] = r[q] = \bar{1}$. Given a vertex $q \in Q$ and an edge $e \in E[q]$, a relaxation step on e is performed by lines 11-13 of the pseudocode. Each time through the **while** loop of lines 5-15, a vertex q is extracted from S (lines 6-7). The value of $r[q]$ just after extraction of q is stored in R, and then $r[q]$ is set to $\bar{0}$ (lines 8-9). Lines 11- 13 relax each edge leaving q. If the tentative shortest distance $d[n[e]]$ is updated during the relaxation and if $n[e]$ is not already in S, the vertex $n[e]$ is inserted in S so that its leaving edges are later relaxed (lines 14-15). $r[n[e]]$ is updated whenever $d[n[e]]$ is, to keep track of the total weight added to $d[n[e]]$ since $n[e]$ was last extracted from S or since the time after initialization if $n[e]$ has never been extracted from S. Finally, line 16 resets the value of $d[i]$ to $\bar{1}$. The algorithm works with any queue discipline chosen for S.

It can be proved that using either tropical or log semiring pushing in the minimization step results in equivalent machines with the same number of states and transitions [28]. However it is radically different on how the weights are distributed along a path. Experimental tests show that pushing in the log semiring benefits speech recognition pruning while using the tropical semiring can, in fact, be harmful in some cases [28]. In our tests pushing in the tropical semiring also gives good results.

5.3.3 Complexity

The complexity of the algorithm depends on the semiring and the choice of queue discipline. Let T_i be the worst cost for inserting a state q in S, T_e be the worst cost for extracting q from S, N_q be the number of times that q has been inserted in S, and $T_{\oplus}$, $T_{\otimes}$, and T_a be the complexity of $\oplus$, $\otimes$, and assignment operation, respectively.

The total complexity, in general, for this algorithm is [32] :

$$O(|Q| + (T_{\oplus} + T_{\otimes} + T_a) \cdot \sum_{q \in Q} (|E[q]| \cdot N_q)) + (T_i + T_e) \cdot \sum_{q \in Q} N_q)$$

5.4 The Automata Minimization

5.4.1 Partitioning

Suppose we are given a set S, and an initial partition π of S into disjoint blocks $\{B_1, B_2, ..., B_p\}$. We are also given a function f on S. Our task is to find the coarsest (having fewest blocks) partition of S, say $\pi'=\{E_1, E_2, ..., E_q\}$, such that:

1. π' is consistent with π (that is, each E_i is a subset of some B_j), and
2. a and b in E_i implies $f(a)$ and $f(b)$ are in some E_j.

We shall call π' the coarsest partition of S compatible with π and f.

The obvious solution is to repeatedly refine the blocks of the original partition by the following method. Let B_j be a block. Examine $f(a)$ for each a in B_i. B_i is then partitioned so that two elements a and b are put in the same block iff $f(a)$ and $f(b)$ are both in some block B_j. The process is iterated until no further refinements are possible. This method yields an $O(n^2)$ algorithm, since each refinement requires time $O(n)$ and there can be $O(n)$ refinements.

We can develop a partitioning algorithm that in refining a block into two subblocks requires time proportional to the smaller subblock. This approach results in an $O(n \log n)$ algorithm [2].

For each $B \subseteq S$, let $f^{-1}(B)=\{b \mid f(b) \in B\}$. Instead of partitioning a block B_i by the values of $f(a)$ for $a \in B_i$, we partition with respect to B_i those blocks B_j which contain at least one element in $f^{-1}(B_i)$ and one element not in $f^{-1}(B_i)$. That is, each such B_j is partitioned into sets $\{b \mid b \in B_j \text{ and } f(b) \in B_i\}$ and $\{b \mid b \in B_j \text{ and } f(b) \notin B_i\}$.

Once we have partitioned with respect to B_i, we need not partition with respect to B_i again unless B_i itself is split. If initially $f(b) \in B_i$ for each element $b \in B_j$, and B_i is split into B_i' and B_i'', then we can partition B_j with respect to either B_i' or B_i'' and we will get the same result since $\{b \mid b \in B_j \text{ and } f(b) \in B_i'\}$ is identical to $B_j - \{b \mid b \in B_j \text{ and } f(b) \in B_i''\}$.

Since we have our choice of partitioning with respect to either B_i' or B_i'', we partition with respect to the easier one. That is, using the smaller of $f^{-1}(B_i')$ and $f^{-1}(B_i'')$. The algorithm is given below:

Input. A set of n elements S, an initial partition $\pi=\{B_1, B_2, ..., B_p\}$. and a function $f: S \to S$.

Output . A partition $\pi'=\{B_1, B_2, ..., B_q\}$ such that π' is the coarsest partition of S compatible with π and f.

Partition algorithm :

1. WAITING$\leftarrow\{1, 2, \ldots, p\}$
2. $q \leftarrow p$;
3. While WAITING not empty do
4. Select and delete any integer i from WAITING;
5. $B_{inv} \leftarrow f^{-1}(B_i)$;
6. For each j such that $B_j \cap B_{inv} \neq 0$, $B_j \not\subset B_{inv}$ do
7. $q \leftarrow q+1$;
8. creat a new block B_q;
9. $B_q \leftarrow B_j \cap B_{inv}$;
10. $B_j \leftarrow B_j - B_q$;
11. If j is in WAITING then add q to WAITING
12. Else
13. If $\|B_j\| <= \|B_q\|$ then
14. Add j to WAITING
15. Else add q to WAITING

Figure 5.6 Partitioning algorithm

5.4.2 Applications of Partitioning Algorithm

For a finite automaton $MA=(\Sigma, Q, q_0, \delta, F)$, δ is a mapping from $Q \times \Sigma$ to Q rather than just a mapping from Q to Q. However we can treat δ as a set $\{\delta_1, \delta_2, \ldots, \delta_p\}$ of functions on S, where each δ_a is the restriction of δ to the input symbol a.

By placing pairs (i, δ_a) in the WAITING. Each pair (i, δ_a) consists of the index i of block a of the partition, plus δ_a, the function on which to partition.

Initially, WAITING = $\{ (i, \delta_a) | \ i = 1 \text{ or } 2 \text{ and } a \in \Sigma\}$, since initial partition $\{ F, Q-F\}$ has two blocks.

Definition 1 : Consider the equivalence relation $\approx$ on Q defined by

$$q \approx p \text{ iff } \forall \varpi \in \Sigma^* : \delta(q, \varpi) \in F \Leftrightarrow \delta(p, \varpi) \in F$$

For $q \in Q$, the equivalence class of Q containing q is denoted by $[q]$. Then the minimal DFA for L is given by $MA=(\Sigma, Q', q_0', \delta', F')$, where

$Q'=Q/\approx$

$q_0'=[q_0]$

$F'=\{[f] | f \in F\}$

$\delta'([q], a)=[\delta(q, a)]$ for all $q \in Q$ and all $a \in \Sigma$

Note that δ is well-defined since for $p, q \in Q$ *with* $p \approx q$ *one has* $\forall a \in \Sigma$: $\delta(p, a)=\delta(q, a)$.

All DFA's that accept the same language as a given minimal DFA are either isomorphic to it or have more states.

Definition 2 : For all $a \in \Sigma$, *let* $\delta_a : Q \to Q$, $q \Rightarrow \delta(q, a)$.

Call a partition π of Q *compatible with* δ, iff

$$\forall C \in \pi \ \forall \varpi \in \Sigma \ \exists B \in \pi : C \subseteq \delta_a^{-1}(B)$$

Lemma 1 : The partition induced by $\approx$ is the coarsest partition compatible with δ that refines $\{F, Q \backslash F\}$.

Lemma 1 allows us to develop an interesting algorithm whose main idea is to refine the initial partition $\{F, Q \backslash F\}$ stepwise until it is compatible.

Let π be a partition of Q that we intend to make compatible by refinement. We call C the elements of π blocks. Let B be such a block and $a \in \Sigma$.

We call a block $C \in \pi$ *compatible wrt* B *and* a iff $C \cap \delta_a^{-1}(B) = 0$ or $C \subseteq \delta_a^{-1}(B)$.

If there is a block C not compatible wrt B and a, then this block C must not stay in the partition because it is not compatible with any refinement of π. So we split C up into C_1 and C_2 with $C_1 \leftarrow C \cap \delta_a^{-1}(B)$, $C_2 \leftarrow C \backslash \delta_a^{-1}(B)$. Afterwards, both C_1, C_2 are compatible wrt B and a, and the new partition π obtained from the old one by $\pi \backslash \{C\} \cup \{C_1, C_2\}$.

We have to repeat this "splitting wrt B and a" until every block is compatible wrt every other block and every letter in Σ.

Once all blocks have been treated wrt B and a (and have been split if necessary), we need not split wrt B and a again until B itself is split. This is true because any subset of a block compatible wrt B and a keeps that property.

The created new blocks C_1 and C_2 might make it necessary to split further blocks (wrt C_i and some $a' \in \Sigma$), so we will have to use some mechanism to remember them.

If C had to be remembered before we split it, we may forget C and concentrate on C_1 and C_2 instead, since any block that is compatible wrt C_1 and C_2 is compatible wrt $C = C_1 \cup C_2$ as well.

If C had not been one of the blocks to remember, this would be due to the fact that all other blocks were compatible to C. We observe that for all blocks D compatible with C, we have that D is compatible with C_1 iff D is compatible with C_2. Therefore it suffices to remember either C_1 or C_2 because splitting wrt one of the C_i will make sure that the resulting blocks are compatible with the other one as well. Of course, it is convenient to choose to remember the smaller of C_1 and C_2.

5.4.3 The Automata Minimization Algorithm

The algorithm is described below. The correctness of the algorithm is proved in [2].

The automata minimization algorithm :

(1) $\pi \leftarrow \{F, Q \backslash F\}$.

(2) *for all* $a \in \Sigma$ *do*

(3) *if* $|F| \leq |Q \backslash F|$ then

(4) $W(a) \leftarrow \{F\}$

(5) else

(6) $W(a) \leftarrow \{Q \backslash F\}$

(7) while $\cup_{a \in \Sigma} W(a) \neq 0$ do

(8) select one $b \in \Sigma$ and delete any B from $W(b)$

(9) $B_{inv} \leftarrow \delta_b^{-1}(B)$

(10) for each $C \in \pi$ not compatible wrt B and b (i.e. $C \cap B_{inv} \neq 0$ and $C \not\subset B_{inv}$) do

(11) $C_1 \leftarrow C \cap B_{inv}$

(12) $C_2 \leftarrow C \backslash B_{inv}$

(13) $\pi \leftarrow \pi \backslash \{C\} \cup \{C_1, C_2\}$

(14) for all $a \in \Sigma$ do

(15) if C is in $W(a)$ then

(16) $W(a) \leftarrow W(a) \backslash \{C\} \cup \{C_1, C_2\}$

(17) else

(18) if $|C_1| < |C_2|$ then

(19) $W(a) \leftarrow W(a) \cup \{C_1\}$

(20) else

(21) $W(a) \leftarrow W(a) \cup \{C_2\}$

Figure 5.7 The automata minimization algorithm

We can extend the automata minimization algorithm to the transducer minimization as follows:

Finite state machine $MA=(\Sigma, Q, q_0, \delta, F)$:

(1) $\pi \leftarrow \{F, \{q_0\}, Q'\}$ where $Q' \leftarrow Q \backslash \{q_0\}$

(2) $Wlist \leftarrow \{F'\}$ where $F' \leftarrow \{(i, b) \mid i \in F \cup \{q_0\}, b \in \Sigma$ and b goes to $i\}$

(3) while $Wlist \neq 0$ do

(4) select one $b \in \Sigma$ *in any* $A' \in Wlist$ *and* delete (i, b) for all $i \in A$ where A is a ***correspondence block*** of A' (see N.B.)

(5) if A' is empty then delete A' from *Wlist*

(6) $B_{inv} \leftarrow \delta_b^{-1}(A)$

(7) for each block/set $C \in \pi$ not compatible wrt block A and edge b (i.e. $C \cap B_{inv} \neq 0$, $C \not\subset B_{inv}$ and $C \neq B_{inv}$) do

(8) $C_1 \leftarrow C \cap B_{inv}$

(9) $C_2 \leftarrow C \backslash B_{inv}$

(10) $\pi \leftarrow \pi \backslash \{C\} \cup \{C_1, C_2\}$

(11) if C', the correspondence block of C, is in *Wlist* then

(12) $Wlist \leftarrow Wlist \backslash \{C'\} \cup \{C_1', C_2'\}$

(13) else

(14) if $|C_1| < |C_2|$ then

(15) $Wlist \leftarrow Wlist \cup \{C_1'\}$

(16) else

(17) $Wlist \leftarrow Wlist \cup \{C_2'\}$

Figure 5.8 The transducer minimization algorithm

N.B. :

(1) The block in π is different from the block in *Wlist* in terms of their elements. That is, the elements in π is a set of states only and in *Wlist* is a set of state and incoming edge pairs. But for each block A' in *Wlist* there is a ***correspondence block*** A in π. In other words, if $A \in \pi$ *i.e.* $A = \{i \mid i \in Q\}$ then there is a correspondence block A' in *Wlist* $A' = \{(i, b) \mid i \in A, b \in \Sigma$ and b goes to $i\}$, e.g. $C' = \{(i,a) \mid i \in C, C \in \pi, a \in \Sigma$ and a goes to $i\}$, $C_1' = \{(i,a) \mid i \in C_1, C_1 \in \pi, a \in \Sigma$ and a goes to $i\}$

$C_2' = \{(i,a) \mid i \in C_2, C_2 \in \pi, a \in \Sigma$ and a goes to $i\}$

For the weighted transducer in the compaction algorithm, the triple string-string/weight is just considered as a unique label, i.e. the unique edge. So each element in Σ is a string-string/weight triple.

5.5. Complexity of Transducer Compaction

In the general case, the classical automata minimization algorithm [2] runs in time $O(|\Sigma| \cdot |Q| \cdot log|Q|)$. It can be shown that a better implementation of the algorithm described in [2] makes it independent of the size of the alphabet. It then depends only on the in-degree of each state. Thus, a better evaluation of the running time of this algorithm is $O(|E| \cdot log\ |Q|)$. Hence the transducer compaction minimization step runs in $O(|E| \cdot log\ |Q|)$.

The overall complexity of the compaction algorithm is analyzed as follows:

(1) if the converted automaton is non-deterministic, the determinization step runs in $O(|Q|2^{|Q|})$ and it dominates the running time of the transducer compaction algorithm. Thus the overall complexity is: $O(|Q|2^{|Q|})$.

(2) If the converted automaton is deterministic, the transducer compaction algorithm can just check the determinism and then bypass the determinization step. The minimization step dominates the running time of the algorithm and it takes $O(|E| \cdot log\ |Q|)$.

5.6 Transducer Compaction in Speech Recognition

In the previous chapter we have shown that the final transducer N can be formed with operations following equation 4.2, i.e. $N = \pi_\varepsilon(\min(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G)))))$. N is not necessarily determinizable, because of the substitutions of distinct auxilliary markers with unique epsilons, and may possibly be reduced furthermore. However, instead of using equation 4.2 we can incorporate the transducer compact operation--*compact* on equation 4.1 directly, i.e.,

$$N = compact(\pi_\varepsilon(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G))))) \qquad (5.1)$$

And then we can proceed with epsilon removal operation rem_ε on it, finally we have the transducer N as:

$$N = rem_\varepsilon(compact(\pi_\varepsilon(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G)))))) \qquad (5.2)$$

Now N is a compact representation of the recognition network as a single transducer and we can use it to build a speech recognizer to run our speech recognition tasks.

In this chapter a transducer compaction operation is introduced. Its application to speech recognition gives a very compact representation of the recognition network and better time and space efficiency.

In the next chapter we will run some tests on the transducers used for building a speech recognizer in CRIM.

Chapter 6

The Experimental Tests of the Transducer Compaction Algorithm

In the last chapter we have introduced a transducer compaction algorithm and its application in speech recognition network. In this chapter we will test the transducer compaction on some transducers currently used in CRIM for building speech recognizers. Test results show that our transducer compaction operation can increase time and space efficiency in speech recognition.

6.1 Test Components

Thanks to Patrick Cardinal who developed most of the transducer operation code for the *FSM* tools in the speech recognition group of CRIM, we can use his *FSMPush*, *FSMEncode*, *FSMDecode*, *FSMDeterminize*, *FSMRmEpsilon*, and *FSMCompact* which was written by me to test the transducer compaction algorithm.

FSMPush implements the weight pushing operation on transducers. The weights are pushed towards the initial state as much as possible.

FSMEncode implements the conversion of a transducer into an automaton and creates an additional symbol transducer (in fact, it is only a mapping list) to remember the mapping of labels in conversion.

FSMDecode implements the recovery of the transducer by decoding back the arcs of the the automaton into original arcs.

FSMDeterminize implements the transducer determinization, it does the same operation as the automata determinization when the transducer is converted as an automaton by using FSMEncode operation.

FSMRmEpsilon removes all the epsilon labels in the transducers.

FSMCompact implements the classical minimization on transducers by considering all labels and weights of one arc as a single label without explicitly encoding them.

6.2 Experimental Tests

The AUPELF task [13] is tested in our experiments.

6.2.1 AUPELF Task

AUPELF is a standard French language, large vocabulary dictation task from the AUPELF 97 evaluation [13], which has a vocabulary of 20,000 words. Acoustic models were trained on BREF-80 and a subset of BREF-Total, containing 53 hours of speech from 100 speakers. Acoustic parameters were 12 Mel-Frequency Cepstral Coefficients (MFCCs) plus energy, and their derivatives. Cross-word triphone acoustic models were trained with 3981 output distributions, each being a Gaussian mixture with 32 components and sharing a single global full covariance matrix. Note that models were not gender-dependent, and that no speaker adaptation (such as VTLN or MLLR) was used.

N.B.: MFCCs is a representation defined as the real cepstrum of a windowed short-time signal derived from the Fast Fourier Transform (FFT) of that signal. The difference from the real cepstrum is that a non-linear frequency scale is used, which approximates the behavior of the auditory system.

6.2.2 Test and Result

The tests are done on a Pentium III 866 MHZ under the Linux operating system with the following command in pipe for transducer compaction operation:

FSMPush –i input_transducer | FSMEncode – key.fst | FSMDeterminize | FSMDecode – key.fst | FSMCompact > output_transducer

key.fst is the additional symbol transducer (in fact, it is only a mapping list) created by FSMEncode operation to remember the mapping of labels in conversion.

Results are given in the following table:

Transducer	HCDG size (states/arcs)	Recognition Time	Recognition Accuracy
$\pi_\varepsilon(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G))))$	6,926,124 /9,971,017	1.42X	84.3%
$compact(\pi_\varepsilon(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G)))))$	6,035,816/9,044,551	1.33X	84.3%
$rem_\varepsilon(compact(\pi_\varepsilon(\det(\tilde{H} \circ \det(\tilde{C} \circ \det(\tilde{L} \circ G))))))$	4,561,788/11,619,818	1.22X	84.3%

Table 6.1 Test result on AUPELF task

N.B.: X stands for the real-time performance.

Recognition accuracy is computed by the following equation:

Accuracy = [1 - (S + I + D)/N)] * 100%

Where S represents substitutions of words by other words, I insertions of words, D deletions of words, and N the total words in the test corpus.

From the results we can see clearly that with transducer compaction operation, both the size of the transducer and the recognition time are reduced. The number of states and arcs are reduced by 13% and 9% respectively; while the recognition time is reduced by 6% with the recognition accuracy being unchanged (84.3%).

Moreover, with further ε-removal operation we obtain even better results than that with transducer compaction. The improvement is also very significant, with the number of states and the recognition time reduced by 24% and 8% respectively. But the number of arcs is increased by 28%, which can cancel the effect of reduction in number of states. However, the recognition accuracy keeps unchanged. In general, it further reduces the recognition time.

The test results show that the transducer compaction operation meets our objective. It can apply on non-determinizable transducers to considerably increase time and space

efficiency. It can be considered as an extension of the minimization operation with better flexibility and efficiency in CSR.

However, the transducer compaction operation requires a lot of computation and memory when the transducers are non-determinizable; while for deterministic transducers it performs as well as minimization.

In this chapter we have described the experimental test and the results. The results meet the objective of our research. Thus the transducer compaction operation can be applied on non-determinizable transducers to reduce its size and acquire time and space efficiency in CSR. In the next chapter we will conclude this research work and present the future work.

Chapter 7
Conclusion

7.1 Review of the Work

This thesis has addressed statistical CSR, several state-of–the-art weighted finite-state transducer algorithms, and their application to speech recognition, and has presented a transducer compaction algorithm for non-determinizable transducers which can be applied in speech recognition for increasing time and space efficiency. In this chapter we summarize this research as follows:

1. The statistical CSR problem is a decoding problem which searches word sentences with the largest probability when an utterance is given. In statistical framework, it is assumed that each word can be expanded in a sequence of context-dependent HMM states, possibly conditioned on the neighboring words in case of cross-word modeling. In practice, the Viterbi criterion is applied, and under this maximum approximation, the search space can be described as a huge network to be explored for finding the best path. The recognized words are then determined by the most probable state sequence. The combinatorial nature of possible state sequences makes the search problem a formidable task, especially for LVCSR. Therefore, time and space requirements are the major concerns for LVCSR, and the decoding strategy is another concern. To ease these problems we resort to the WFST approach in CSR.

2. In the WFST approach, all components used in the search stage of CSR system –language model, pronunciation dictionary, phonetic context-dependency, HMM model – are represented as WFSTs. These individual models are then combined and optimized using the general weighted finite-state operations of composition, determinization, compaction, and weight "re-distribution" ("pushing"). The purpose of these steps is to create a single optimized transducer that maps directly sequences of HMM-state-level distributions to sequences of words. Since the search transducer is built statically, these optimizations can be performed entirely off-line.

In WFSTs, the components of the recognition network, the transducers, are handled in a highly flexible way, independently of the decoder specifics about contextual range. The

final optimized network is substantially reduced to only twice that of language model [27]. Cross-word context expansion increases the network by just a few percents with respect to the optimized context-independent network [27].

3. The composition, determinization, and minimization algorithms remove redundancy and minimize the size of the recognition transducer in order to increase time and space efficiency for CSR.

- Composition: Composition is the key operation on transducers in speech recognition. It can construct complex transducers from simpler ones. Also it is useful for combining different levels of representation in speech recognition. Thus composition makes it possible to combine all transducers in cascade of the recognition network and to create an integrated single transducer.
- Determinization: Determinization eliminates redundant paths in the composed network to reduce recognition time. It can also be used in intermediate steps of constructing a minimal transducer in order to improve the efficiency of composition and to reduce the network size. It is mainly used to increase the time efficiency.
- Minimization: Minimization minimizes the size of the transducer considered. It pushes the weights towards the initial states as much as possible, then finds the equivalent states in the transducer, and merges them into one state. It can increase space and time efficiency in CSR.
- Weight pushing: Weight pushing pushes the weights of weighted acceptors or transducers towards the initial states as much as possible. It can be used in intermediate steps of constructing a minimal transducer. Also it can affect the pruning of the decoder. Pushing in the log semiring benefits speech recognition pruning. Thus it can improve the decoding speed.

4. Finally a transducer compaction algorithm is presented. It is designed to reduce the size of non-determinizable transducers. The transducer compaction operation has the following six steps:

1. Weight pushing: to push the weight towards the initial state as much as possible for weighted transducer or acceptor.
2. Encoding: to convert a transducer into an automaton, that is to encode each triple of an input label, output label and cost (weighted transducers) or each double of an input and cost or output (weighted acceptors or string-to-string transducers) into a single new label.
3. Determinization checking: to check the determinism of the new automaton. If it is deterministic skip the determinization step, i.e. step 4, and go to minimization directly.
4. Determinization: to apply the classical automata determinization on the new automaton..
5. Minimization: to apply classical automata minimization on the deterministic automata.
6. Decoding: the encoded labels are decoded back into their original values.

7.2 Future Work

The decoder has a separate representation for variable-length left-to-right HMMs for efficiency reasons, which is called the *HMM specificatio*n. However, the integrated network we have described does not take good advantage of this since, having combined the HMMs into the recognition network proper, the HMM specification consists of trivial one-state HMMs. Thus, by suitably *factoring* the integrated network, we can again take good advantage of this feature. Therefore a suitable factoring can lead to a substantial reduction in the size of the network. Currently a simple factoring method has been found [29]. In the future work we have to find a more effective, less constrained factoring method that can be used in CSR.

References

1. A. Salomma. *Formal Languages*. Academic Press, New York, NY. 1973.
2. A.V. Aho, John E. Hopcroft, and J. D. Ullman. "*The Design and Analysis of Computer Algorithms*". Addison Weley: Reading, MA. 1974.
3. A.V. Aho, R. Sethi, and J. D. Ullman. "*Compilers, Principles, Techniques and Tools*". Addison Wesley: Reading, MA. 1986.
4. D. Perrin. Finite Automata. In J. Van Leuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 1-57. Elsevier, Amsterdam, 1990.
5. D. Revuz. "Minimisation of Acyclic Deterministic Automata in Linear Time". *Theoretical Computer Science*, 92(1): 181-189, 1992.
6. E. Roche and Y. Schabes. "*Finite-State Language Processing*". A Bradford Book, The MIT Press, Cambridge, Massachusetts. London England. 1997.
7. F. C. N. Pereira and M. D. Riley. "Speech Recognition by Composition of Weighted Finite Automata". *Finite-State Language Processing*, edited by Emmanuel Roche and Yves Schabes. A Bradford Book, The MIT Press, Cambridge, Massachusetts. London England. 1997.
8. F. Jelinek. "Continuous speech recognition by statistical methods". *Proceedings of the IEEE*, 64:532-556, 1976.
9. H. Ney, U. Essen, and R. Kneser. On Structuring Probabilistic Dependences in Stochastic Language Modelling. *Computer Speech and Language*, 8:1-38, 1994.
10. J. Berstel. *Transductions and Context-Free Languages*. B. G. Teubner Stuttgart, 1979.
11. J. Berstel and C. Reutenauer. *Rational Series and Their Language*s. Springer-Verlag: Berlin-New York, 1988.
12. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley: Reading, MA, 1979

13. J.M. Dolmazon, F. Bimbot, G. Adda, M. El Beze, J.C. Caerou, J. Zeiliger, and M. Adda-Decker. ARC B1 - Organisation de la première campagne AUPELF pour l'évaluation des systèmes de dictée vocale, JST97 FRANCIL, Avignon, Avril 1997.
14. J. Qiu. *Determinization of String-to-String/Weight Finite State Transducers in Speech Recognition*. Master's thesis, McGill University, School of Computer Science, 2000.
15. K. Koskenniemi. "Finite-State Parsing and Disambiguation". In *proceedings of the 13th International Conference on Computational Linguistics. COLING-90*, Vol. 2. Helsinki, Finland, pages 229-232, 1992.
16. K. Koskenniemi, P. Tapanainen and A. Voutilainen. "Compiling and Using Finite-State Syntactic Rules". In proceedings of the 15th International Conference on Computational Linguistics. COLING-92, Vol. I., pages 156-162, Nantes, France. 1992.
17. M. Mohri. On Some Applications of Finite-State Automata Theory to Natural Language Processing. *Natural Language Engineering*, 2:1-20, 1996.
18. M. Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23:2, pages 269-312, 1997.
19. M. Mohri. "On the Use of Sequential Transducers in Natural Language Processing". *Finite-State Language Processing*, edited by E. Roche and Y. Schabes. A Bradford Book, The MIT Press, Cambridge, Massachusetts. London England, pages 355-382, 1997.
20. M. Mohri. Minimization Algorithms for Sequential Transducers. *Theoretical Computer Science*, 234(1-2): 177-201, 2000.
21. M. Mohri. Compact Representations by Finite-State Transducers. *Proceedings of the 32nd Meeting of the Association for Computational Linguistics (ACL 94)*, pages 204-209, Las Cruces, New Mexico, 1994.
22. M. Mohri and M. Riley. "Weighted Determinization and Minimization for Large Vocabulary Speech Recognition". In Proceedings of the Eurospeech '97, pages 131-134, Rhodes, Greece, 1997.

23. M. Mohri. General Algebraic Frameworks and Algorithms for Shortest-Distance Problems. *Technical Memorandum 981210-10TM*, AT&T Labs - Research, 62 pages, 1998.
24. M. Mohri. *Speech Processing*. Graduate course, Columbia University, Department of Computer Science, New York, NY, 515 pages. 1998.
25. M. Mohri, F. C. N. Pereira, and Michael Riley. Weighted Automata in Text and Speech Processing, In *ECAI-96 Workshop*, pages 46-50, Budapest, Hungary, 1996
26. M. Mohri, F. C. N. Pereira, and M. Riley. The Design Principles of a Weighted Finite-state Transducer Library. *Theoretical Computer Scienc*e, 231:17–32, January 2000.
27. M. Mohri and Michael Riley. Integrated Context-Dependent Networks in Very Large Vocabulary Speech Recognition. In *Proceedings of the 6th European Conference on Speech Communication and Technology (Eurospeech '99)*. Volume 2, 811-814, Budapest, Hungary, 1999.
28. M. Mohri and M. Riley. A Weight Pushing Algorithm for Large vocabulary Speech Recognition. In *(Eurospeech '01)*. Pages 70-73, Aalborg, Denmark, September 2001.
29. M. Mohri, F.C.N. Pereira, and M. Riley. Weighted Finite-State Transducers in Speech Recognition. *Computer Speech and Language*, 16(1):69-88, 2002.
30. M.O. Rabin and D. Scott. Finite Automata and their decision problems. *IBM J. Res and Develop*. 3:2(1959) 115-125.
31. M. Siper. *Introduction to the Theory of Computation*. PWS Publishing Company, 20 Park Plaza. Boston, MA 02116. 1997.
32. P. Cardinal. *Finite-State Transducers and Speech Recognition*. Master's thesis, McGill University, School of Computer Science, 2002.
33. R.M. Kaplan and M. Kay. "Regular Models of Phonological Rule Systems". *Computational Linguistics*, 20, 1994.
34. S.C. Kleene. "Representation of Events in Nerve Nets and Finite Automata". In C. E. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press. 1956.
35. S. Eilenberg. *Automata, Languages, and Machines*. Volume A. Academic Press, New York. 1974.

36. S. Eilenberg. *Automata, Languages, and Machines*. Volume B. Academic Press, New York. 1976.
37. S.J. Young. "Statistical Modelling in Continuous Speech Recognition." *Proc Int. Conference on Uncertainty in Artificial Intelligence*, pages 562-571, Seattle, WA, August 2001.
38. S. Katz. "Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer", *IEEE Trans. Acoustics Speech and Signal Processing*, 35, 400–401, 1987.
39. S. L. Graham, M. A. Harrison, and W.L. Ruzzo. "An Improved Context-Free Recognizer". *ACM Transactions on Programming Languages and Systems*, 2, 1980.
40. S. Ortmanns, H. Ney and A. Eiden. Language-model look-ahead for large vocabulary speech recognition. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP'96)*, pages 2095–2098. University of Delaware and Alfred I. duPont Institute, 1996.
41. T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, The MIT Press: Cambridge, MA, 1992.
42. *The HTK Book* (for HTK Version 3.0). Cambridge University, 2000.
43. W. Bauer. "On Minimizing Finite Automata". *SATACS Bulletin*, 35, 1988.
44. W. Kuich and A. Salomaa. Semirings, Automata, Languages. *Number 5 in EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, Germany, 1986.