Weighted Finite-State Transducers in Automatic Speech Recognition

Dr. John McDonough

Spoken Language Systems
Saarland University

January 14, 2010

Introduction

- In this lecture, we will discuss the application of weighted finite-state transducers in automatic speech recognition (ASR).
- We will begin with an examination of conventional finite-state automata, and the power set construction.
- Then we will consider some conventional graph search algorithms.
- Next we will define weighted finite-state transducers (WFSTs), and generalize the power set construction to weighted determinization.
- We will demonstrate how the knowledge sources used to build an ASR search graph can be represented as WFSTs.
- The Viterbi algorithm will be presented as the search for the shortest path through a directed acyclic graph.
- Finally, we will present how context free grammars can be approximated with finite-state automata.

A Man, Goat, Wolf, and a Cabbage

- Problem: A man has a boat, a goat, a wolf, and a cabbage.
- In the boat there is only enough room for the man and one of his possessions.
- The man must cross a river with his entourage but:
- For obvious reasons, the wolf cannot be left alone with the goat.
- For obvious reasons, the goat cannot be left alone with the cabbage.
- How can the man solve this problem?

Finite-State Model

- Luckily, the man has a day job as a professor of computer science, and hence decides to model the task of crossing the river with his property as a finite-state automaton.
- Let the current state of the system correspond to the objects still on the left bank of the river.
- The initial state is MWGC.
- Every time the man crosses the river alone, the transition is labeled with *m*.
- Every time the man crosses the river and carries one of his entourage, the transition is labeled with one of *w*, *g*, or *c*.

Finite-State Model

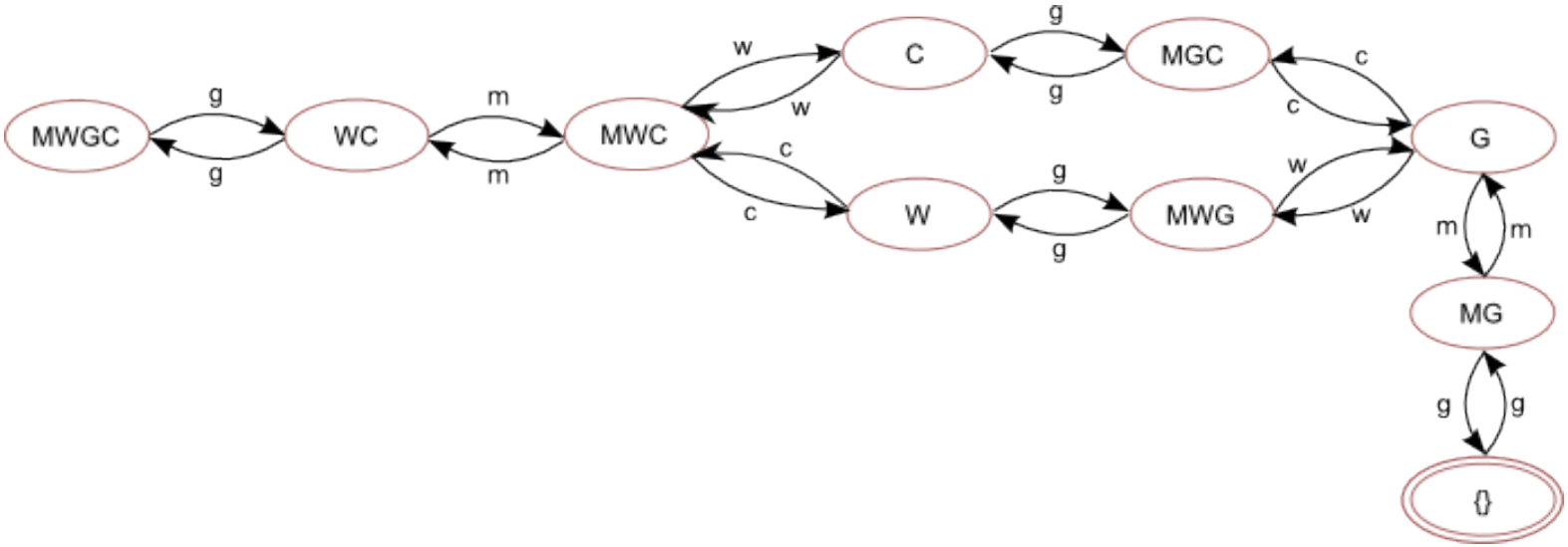


Figure: Transition diagram of the Man-Wolf-Goat-Cabbage problem.

Characteristics of the MWGC-Problem

- A solution to the man-wolf-goat-cabbage problem corresponds to a path through the transition diagram from the start state MWGC to the end state $\emptyset$.
- It is clear from the transition diagram that there are two equally short solutions to the problem.
- All other solutions, of which there infinitely many, involve useless cycles.
- As with all finite-state automata, there is a unique start state.
- This particular FSA also has a single valid end or accepting state, which is not generally the case.

Formal Definitions

Formally define a finite-state automaton (FSA) as the 5-tuple $(Q, \Sigma, \delta, i, F)$ where

- Q is a finite set of states,
- Σ is a finite alphabet,
- $i \in Q$ is the initial state,
- $F \subseteq Q$ is the set of final states,
- δ is the transition function mapping $Q \times \Sigma$ to Q, which implies $\delta(q, a)$ is a state for each state q and input a provided that a is accepted when in state q.

Extending δ to Strings

- To handle strings, we must extend δ from a function mapping $Q \times \Sigma$ to Q, to a function mapping $Q \times \Sigma^*$ to Q, where Σ^* is the Kleene closure of Σ.
- Let $\hat{\delta}(q, w)$ be the state that the FSA is in after beginning from state q and reading the input string w.
- Formally, we require:
  1. $\hat{\delta}(q, \epsilon) = q$.
  2. For all strings w and symbols a, $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$.
- Condition 1) implies that the FSA cannot change state without receiving an input.
- Condition 2) tells us how to find the current state after reading a nonempty input string wa; i.e., find $p = \hat{\delta}(q, w)$, then find $\delta(p, a)$.
- As $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \epsilon), a) = \delta(q, a)$ we will use δ to represent both δ and $\hat{\delta}$ henceforth.

Regular Languages

- A string x is accepted by a FSA $M = (Q, \Sigma, \delta, i, F)$ if and only if $\delta(i, x) = p$ for some $p \in F$.
- The language accepted by M, which we denote as $L(M)$, is that set $\{x | \delta(i, x) \in F\}$.
- A language is a regular set, or simply regular, if it is the set accepted by some automaton.
- $L(M)$ is the complete set of strings accepted by M.

Nondeterministic Finite-State Automata

- Consider a modification to the original definition of the FSA, whereby zero, one, or more transitions from a state with the same symbol are allowed.
- This new model is known as the nondeterministic finite-state automaton (NFSA).
- Observe that there are two edges labeled 0 out of state q_0, one going back to state q_0 and one to state q_1.

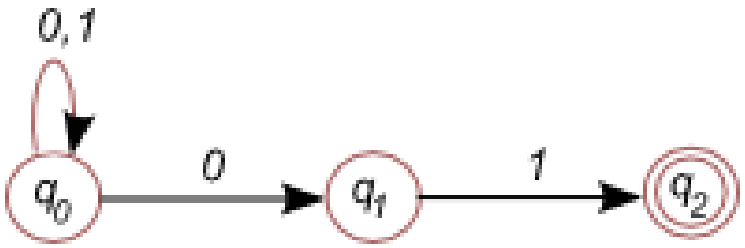


Figure: A nondeterministic finite-state automaton.

Formal Definitions: NFSA

- Formally define a nondeterministic finite-state automaton (NFSA) as the 5-tuple $(Q, \Sigma, \delta, i, F)$ where Q is a finite set of states,
- Σ is a finite alphabet,
- $i \in Q$ is the initial state,
- $F \subseteq Q$ is the set of final states, δ is the transition function mapping $Q \times \Sigma$ to 2^Q, the power set of Q.
- This implies $\delta(q, a)$ is the set of all states p such that there is a transition labeled a from q to p.

Equivalence of DFSAs and NFSAs

Theorem (equivalence of DFSAs and NFSAs): Let L be the set accepted by a nondeterministic finite-state automaton. Then there exists a deteriministic finite-state automaton that accepts L.

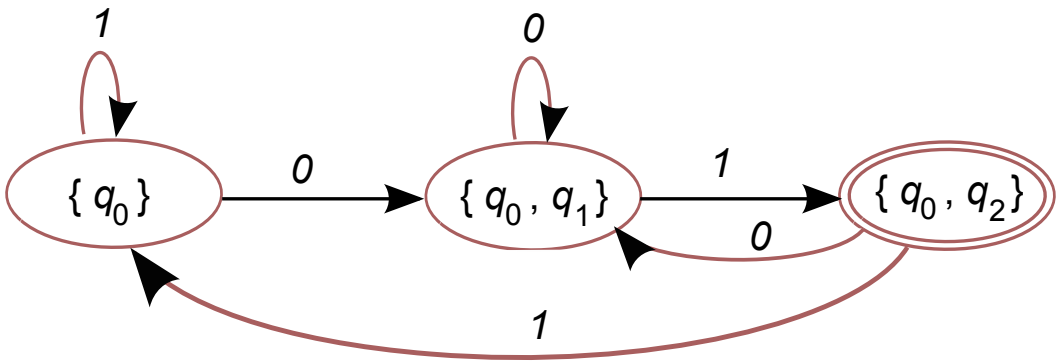


Figure: An equivalent deterministic finite-state automaton.

Power Set Construction

- Let $M_1 = (Q_1, \Sigma, \delta_1, i_1, F_1)$ denote the NFSA accepting L.
- Define a DFSA $M_2 = (Q_2, \Sigma, \delta_2, i_2, F_2)$ as follows:
- The states of M_2 are all subsets of the states of M_1, that is $Q_2 = 2^Q$.
- M_2 keeps track in its states the subset of states that M_1 could be in at any given time.
- F_2 is the subset of states in Q_2 which contain a state $f \in F_1$.
- An element of $m \in Q_2$ will be denoted as $m = [m_1, m_2, \ldots, m_N]$, where each $m_n \in Q_1$.
- Finally, $i_2 = [i_1]$.

Definition of $\delta_2([p_1, p_2, \ldots, p_N], a)$

By definition,

$$\delta_2([m_1, m_2, \ldots, m_N], a) = [p_1, p_2, \ldots, p_N]$$

if and only if

$$\delta_1(m_1, m_2, \ldots, m_N, a) = \{p_1, p_2, \ldots, p_N\}.$$

In other words, $\delta_2([m_1, m_2, \ldots, m_N], a)$ is computed for $[m1, m2, \ldots, mN] \in Q_2$ by applying δ to each $m_n \in Q_1$.

Proof by Induction

We wish to demonstrate through induction on the string length $|x|$ that

$$\delta_2(i_2, x) = [m_1, m_2, \ldots, m_N]$$

if and only if

$$\delta_1(i_1, x) = \{m_1, m_2, \ldots, m_N\}.$$

- Basis: The result follows trivally for $|x| = 0$, as $i_2 = [i_1]$ and $x = \epsilon$.
- Inductive Hypothesis: Assume that the hypothesis is true for strings of length N or less, and demonstrate it is then necessarily true for strings of length $N + 1$.

Proof of Inductive Hypothesis

- Let xa be a string of length $N+1$, where $a \in \Sigma$.
- Then,

$$\delta_2(i_2, xa) = \delta_2(\delta_2(i_2, x), a).$$

- By the inductive hypothesis,

$$\delta_2(i_2, x) = [m_1, m_2, \ldots, m_N]$$

if and only if

$$\delta(i_1, x) = \{m_1, m_2, \ldots, m_N\}.$$

Graph Searches

- The most basic operation on a graph is to search through it to discover all vertices.
- The vertices are assigned a color during the search:
  - A node v that has not been previously discovered is *white*.
  - A node v that has been discovered, but whose edge list has not been fully explored is *gray*.
  - After the adjacency list of v has been fully explored, it is *black*.

  The distance $v.\text{dist}$ of a node v is the number of edges traversed from the initial state i in order to reach v.
- The predecessor $v.\pi$ of a node v is the node from whose edge list v was discovered.

Prof. Bumstead in Disarray

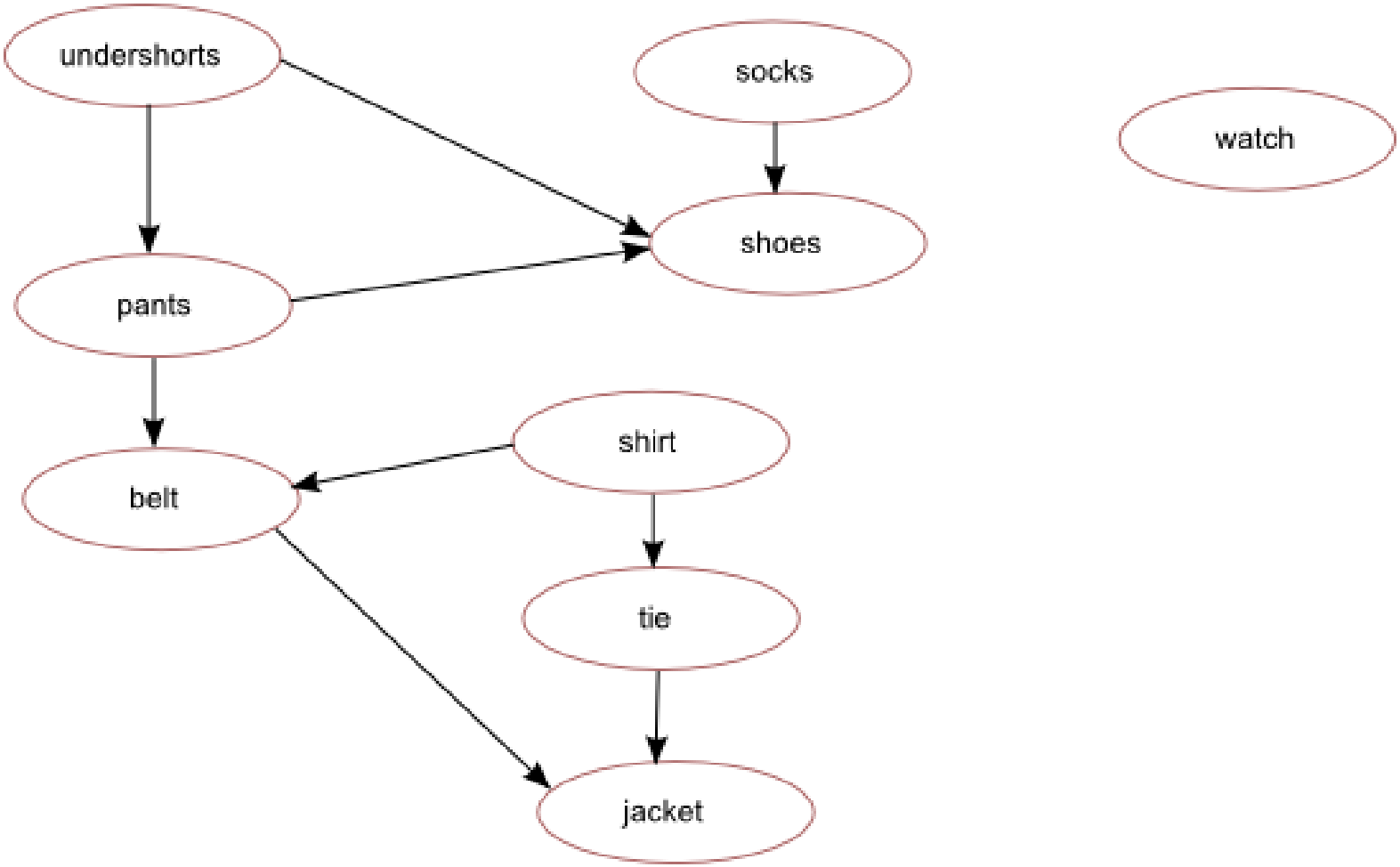


Figure: Prof. Bumstead's dependency diagram.

Alles in Ordnung bei Prof. Bumstead

Figure: Prof. Bumstead's dressing order.

Recursive Function visit(u)

- Assume we have a directed graph $G = (V, E)$ where every $v \in V$ is initially white, and let time denote a global time stamp.
- Define the recursive function visit(u) for some $u \in V$.

```
def visit(u):
    u.color ← Gray # u has been discovered
    u.disc ← time ← time + 1
    for v in u.adj: # explore edge (u, v)
        if v.color == White:
            v.π ← u
            visit(v)
    u.color ← Black # u is done, paint it black
    u.fin ← time ← time + 1
```

Depth First Search

Then the complete depth first search over G can be described as

```
def dfs(G):
    for u in G.V:
        u.color ← White
        u.π ← Null
    time ← 0
    for u in G.V:
        if u.color == White:
            visit(u)
```

Topological Sort

- Let us define a directed acyclic graph (dag) $G = (V, E)$ as a digraph that contains no cycles.
- A *topological sort* is a linear ordering of all $v \in V$ such that if $u \rightarrow v \in E$, then u appears before v in the ordering.
- A topological sort can be performed with the following steps:
  - Call DFS(G) to determine the finishing times v.finish for each $v \in V$.
  - As each v is finished, insert it into the front of a linked list.
  - Upon termination, the linked list contains the topologically sorted vertices.

Object of a Shortest Path Algorithm

- Upon termination, a shortest path algorithm will have set the predecessor $\pi[v]$ for each $v \in V$ such that it points towards the prior vertex on the shortest path from s to v.
- Note that $\pi[v]$ will *not* necessarily point to the predecessor of v on the shortest path from s to v while the algorithm is still running.
- Let us define the *predecessor subgraph* $G_\pi(V_\pi, E_\pi)$ as that graph induced by the back pointers π of each vertex.
- Let us define the set $V_\pi \triangleq \{v \in V : \pi[v] \neq \texttt{NULL}\} \cup \{s\}$.
- The directed edge set E_π is the set of edges induced by the π values for vertices in V_π:

$$E_\pi = \{(\pi[v], v) \in E : v \in V - \{s\}\}.$$

Initialization

- During the execution of a shortest-paths algorithm, we maintain for each $v \in V$ an attribute $d[v]$ which is the current estimate of the shortest path distance.
- The attributes $\pi[v]$ and $d[v]$ are initialized as in the algorithm shown below.
- After initialization, $\pi[v] =$NUL for all $v \in V$, $d[s] = 0$ and $d[v] = \infty$ for $v \in V - \{s\}$.

```
def Initialize-Single-Source(G, s):
    for v ∈ G:
        d[v] ← ∞
        π[v] ← NULL
    d[s] ← 0
```

Relaxation

- The process of *relaxing* an edge $u \rightarrow v$ means testing whether the distance from s to v can be reduced by traveling over u.
- This process is illustrated in the pseudocode given below.
- The relaxation procedure may decrease the value of the shortest path estimate $d[v]$ and update $\pi[v]$.
- The estimate $d[v]$ can never increase during relaxation, only remain the same or decrease.

```
def Relax(u, v, w):
    if d[v] > d[u] + w(u,v):
        d[v] ← d[u] + w(u, v)
        π[v] ← u
```

Single-Source Shortest Paths in DAGs

1. Shortest distances are always well defined in dags (directed acylic graphs), as no negative weight cycles can exist even if there are negative weights on some edges.
2. For a DAG $G = (V, E)$, the shortest paths to all nodes can be found in $\mathcal{O}(V + E)$ time.
3. First the vertices must be topologically sorted.
4. Thereafter the edges from each node can be relaxed, where the vertices are taken in topological order.

```
def DAG-Shortest-Paths(G, w, s):
    sorted = Topo-Sort(G)
    Initiliaze-Single-Source(G, s)
    for u in sorted:
        for v in Adj[u]:
            Relax(u, v, w)
```

Weighted Finite-State Acceptor

- A *weighted finite-state acceptor* is so-named because it *accepts* strings from Σ^*, and assigns a weight to each accepted string.
- A string s is accepted by A iff there is a successful path π labeled with s through A.
- The label $l[\pi]$ for an entire path $\pi = e_1 \cdots e_K$ can be formed through the concatenation of all labels on the individual transitions:
$$l[\pi] \triangleq l[e_1] \cdots l[e_K].$$
- The weight $w[\pi]$ of a path π can be represented as
$$w[\pi] \triangleq \lambda \otimes w[e_1] \otimes \cdots \otimes w[e_K] \otimes \rho(n[e_K]),$$
where $\rho(n[e_K])$ is the final weight.
- Typically, Σ contains ϵ, which denotes the null symbol.
- Any transition in A with the label ϵ consumes no symbol from Σ when taken.

Successful Path

- In the coming development, we will require a formal definition of a *path*:
  A path π through an acceptor A is a sequence of transitions $e_1 \cdots e_K$, such that

$$n[e_k] = p[e_{k+1}] \;\; \forall\, k = 1, \ldots, K-1.$$

- A *successful path* $\pi = e_1 \cdots e_K$ is a path from the initial state i to an end state $f \in F$.

Weighted Finite-State Transducers

- We now generalize our notion of a WFSA in order to consider machines that translate one string of symbols into a second string of symbols from a different alphabet along with a weight.
  Weighted finite-state transducer: A WFST $T = (\Sigma, \Omega, Q, E, i, F, \lambda, \rho)$ on the semiring $\mathbb{K}$ consists
  - of an *input alphabet* Σ,
  - an *output alphabet* Ω,
  - a set of states Q,
  - a set of transitions $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Omega \cup \{\epsilon\}) \times \mathbb{K} \times Q$
  - a initial state $i \in Q$ with weight λ,
  - a set of final states $F \subseteq Q$,
  - and a function ρ mapping from F to $\mathbb{R}^+$.

WFST Transition

A transition $e = (p[e], l_{\mathrm{i}}[e], l_{\mathrm{o}}[e], w[e], n[e]) \in E$ consists of

- a previous state $p[e]$,
- a next state $n[e]$,
- an input symbol $l_{\mathrm{i}}[e]$,
- an output symbol $l_{\mathrm{o}}[e]$, and
- a weight $w[e]$.

Weighted Finite-State Transducers (cont'd.)

- A WFST, such as that shown in Figure 6 maps an input string to an output string and a weight.
- For example, such a transducer would map the input string "red white blue" to the output string "yellow blue red" with a weight of $0.5 + 0.3 + 0.2 + 0.8 = 1.8$.
- It differs from the WFSA only in that the edges of the WFST have *two* labels, an input and an output, rather than one.
- A string s is accepted by a WFST T iff there is a successful path π labeled with $l_{\mathrm{i}}[\pi] = s$.
- The weight of this path is $w[\pi]$, and its output string is

$$l_{\mathrm{o}}[\pi] \triangleq l_{\mathrm{o}}[e_1] \cdots l_{\mathrm{o}}[e_K].$$

- Any ϵ–symbols appearing in $l_{\mathrm{o}}[\pi]$ can be ignored.

Diagram of a Weighted Finite-State Transducer

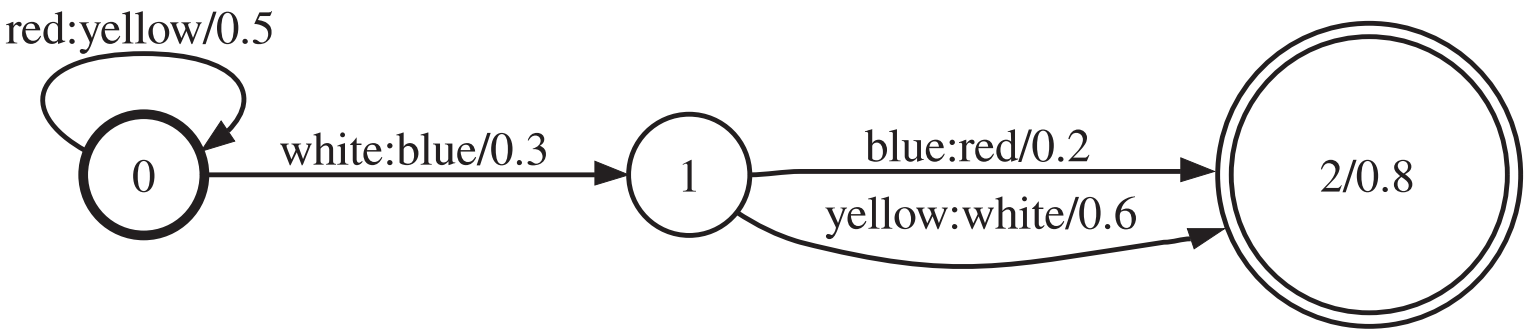


Figure: A simple weighted finite-state transducer.

Weighted Composition

- Consider a transducer S which maps an input string u to an output string v with a weight of w_1, and a transducer T which maps input string v to output string y with weight w_2.
- The *composition*

$$R = S \circ T$$

of S and T maps string u directly to y with weight

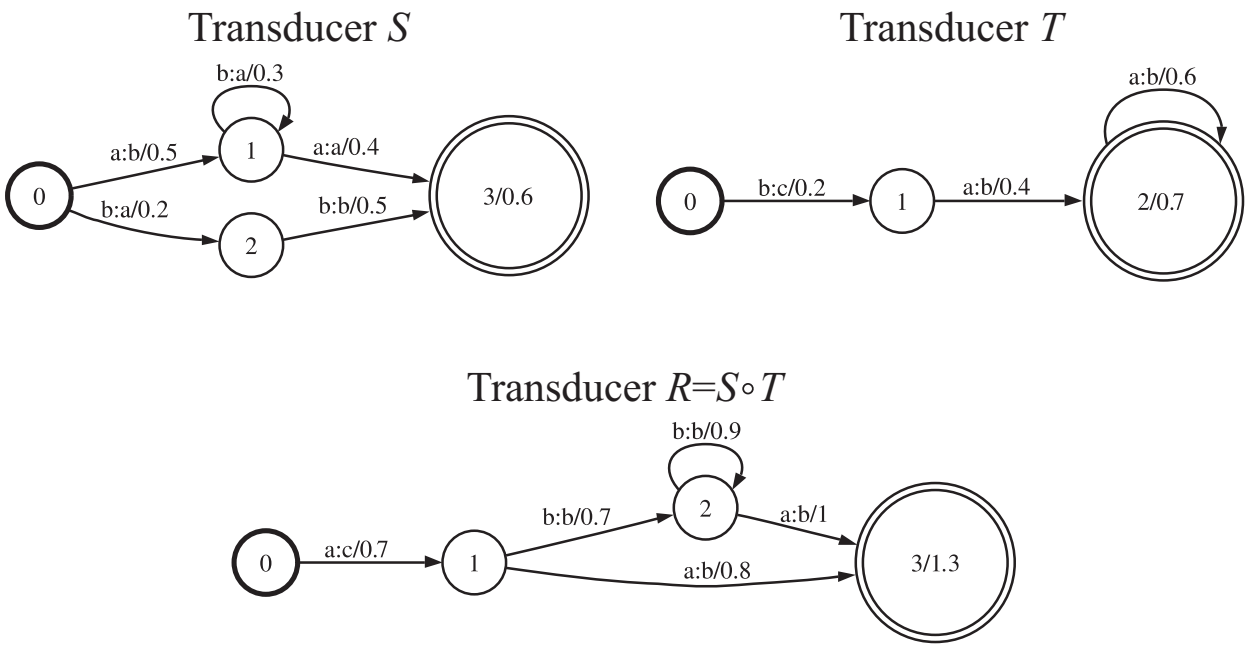
$$w = w_1 \otimes w_2.$$

- We will adopt the convention that the components of particular transducer are denoted by subscripts; e.g., Q_R denotes the set of states of the transducer R.

Weighted Composition (cont'd.)

- In the absence then of ϵ–transitions, the construction of such a transducer R is straightforward.
- The output symbols on the transitions of a node $n_S \in Q_S$ are paired with the input symbols on the transitions of a node $n_T \in Q_T$, beginning with the initial nodes i_S and i_T.
- Each $n_R \in Q_R$ is uniquely determined by the pair (n_S, n_T).

Schematic of Weighted Composition



- The transition from State 0 labeled with a:b/0.5 in S has been paired with the transition from State 0 labeled with b:c/0.2 in T, resulting in the transition labeled a:c/0.7 in R.
- Each new node $n_R = (n_S, n_T)$ is placed on a queue to eventually have its adjacency list expanded.

Weighted Determinization

Definitions:
Equivalent: Two WFSAs are *equivalent* if for any accepted input sequence, they produce the same weight. Two WFSTs are equivalent if for any accepted input sequence they produce the same output sequence and the same weight.
Deterministic: A WFST is *deterministic* if at most one transition from any node is labeled with any given input symbol.

Effects of Weighted Determinization

- The two WFSTs in the figure are equivalent over the tropical semiring in that they both accept the same input strings, and for any given input string, produce the same output string and the same weight.
- For example, the original transducer will accept the input string *aba* along either of two successful paths, namely, using the state sequence $0 \rightarrow 1 \rightarrow 3 \rightarrow 3$ or the state sequence $0 \rightarrow 1 \rightarrow 4 \rightarrow 3$.
- Both sequences produce the string *ab* as output, but the former yields a weight of $0.1 + 0.4 + 0.6 = 1.1$, while the latter assigns a weight of $0.1 + 0.3 + 0.5 = 0.9$.
- Hence, given that these WFSTs are defined over the tropical semiring, the final weight assigned to the input *aba* is 0.9, the minimum of the weights along the two successful paths.

Diagram of Weighted Determinization

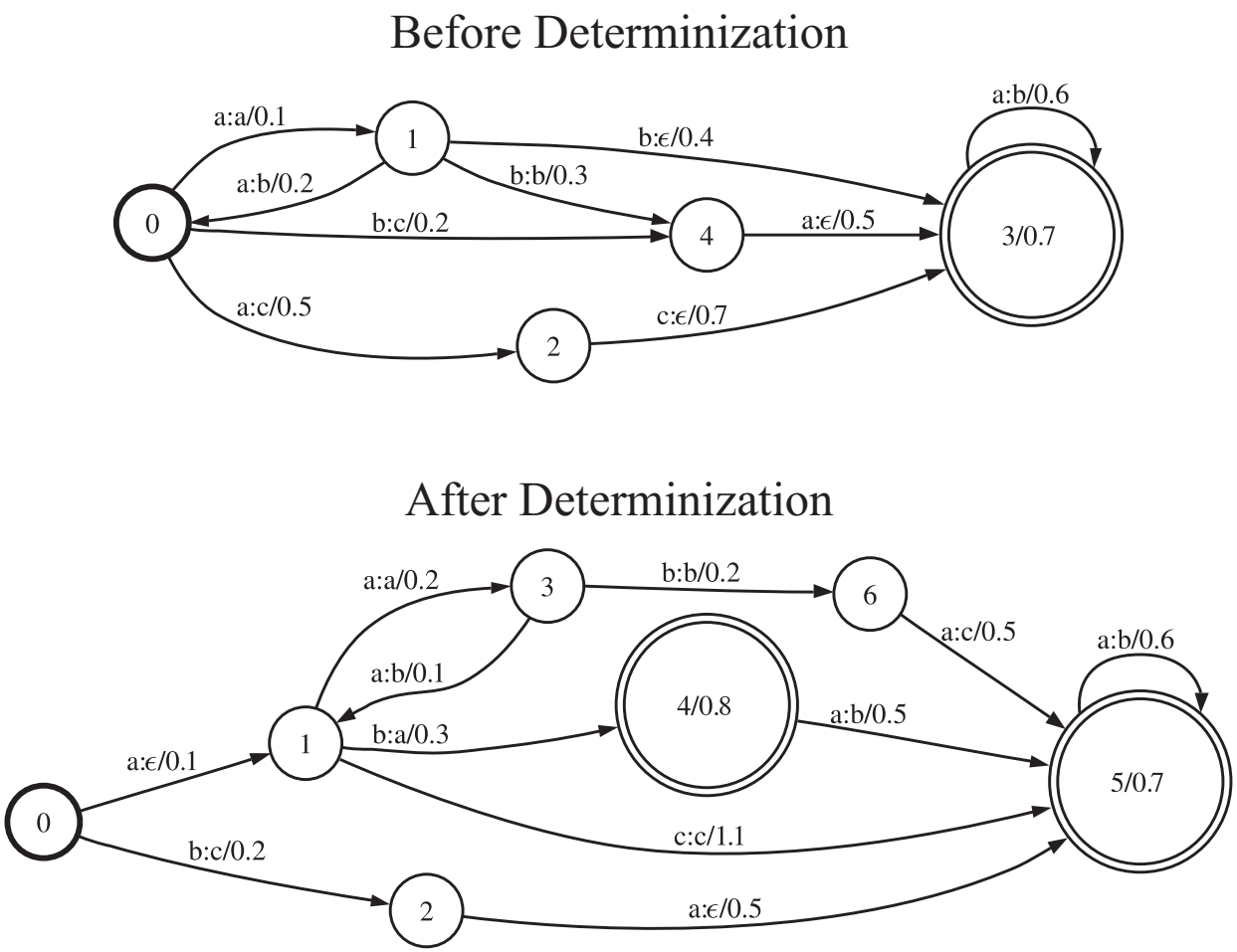


Figure: Weighted determinization of a transducer.

Finite-State Grammar

- The first knowledge source is the *grammar*, of which there are two primary sorts:
  A *finite-state grammar* (FSG) is crafted from rules or other expert knowledge, and typically only accepts a very restricted set of word sequences.
- Such a grammar is illustrated in the figure for a hypothetical travel assistance application.
- The FSG in the figure accepts strings such as “SHOW ME THE QUICKEST WAY FROM ALEWIFE TO BOSTON.”
- FSGs are useful in that their constrained languages help prevent recognition errors.
- Unfortunately, these constrained languages also cause all formulations of queries, responses, or other verbal interactions falling outside of the language to be rejected or misrecognized.

Finite-State Grammar Schematic

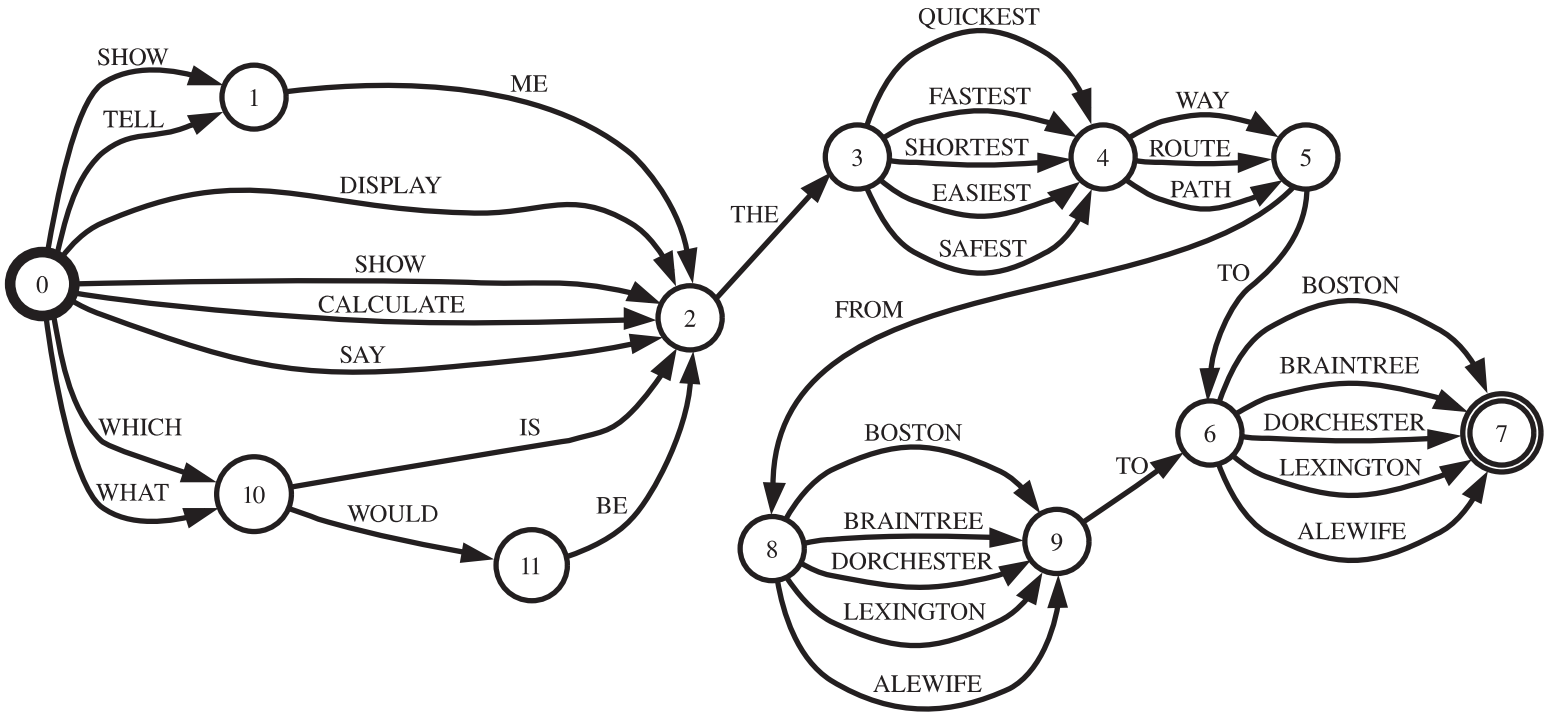


Figure: Finite-state grammar for a hypothetical travel assistance application.

Statistical Language Models

- The second type of grammar is a *statistical language model* or *N–gram*, which assigns negative log-probabilities to sequences of words.
- The *N*–gram typically includes a *backoff* node, which enables it to accept *any* sequence of words.
- The structure of the *bigram*, in which the probability of the current word w_k is conditioned solely on the prior word w_{k-1}, is shown below.
- From the figure, it is clear that the bigram contains two kinds of nodes.
- The first type is the actual bigram node, for which all incoming transitions must be labeled with the same prior word w_{k-1} in order to uniquely specify the context.

N-Gram Schematic

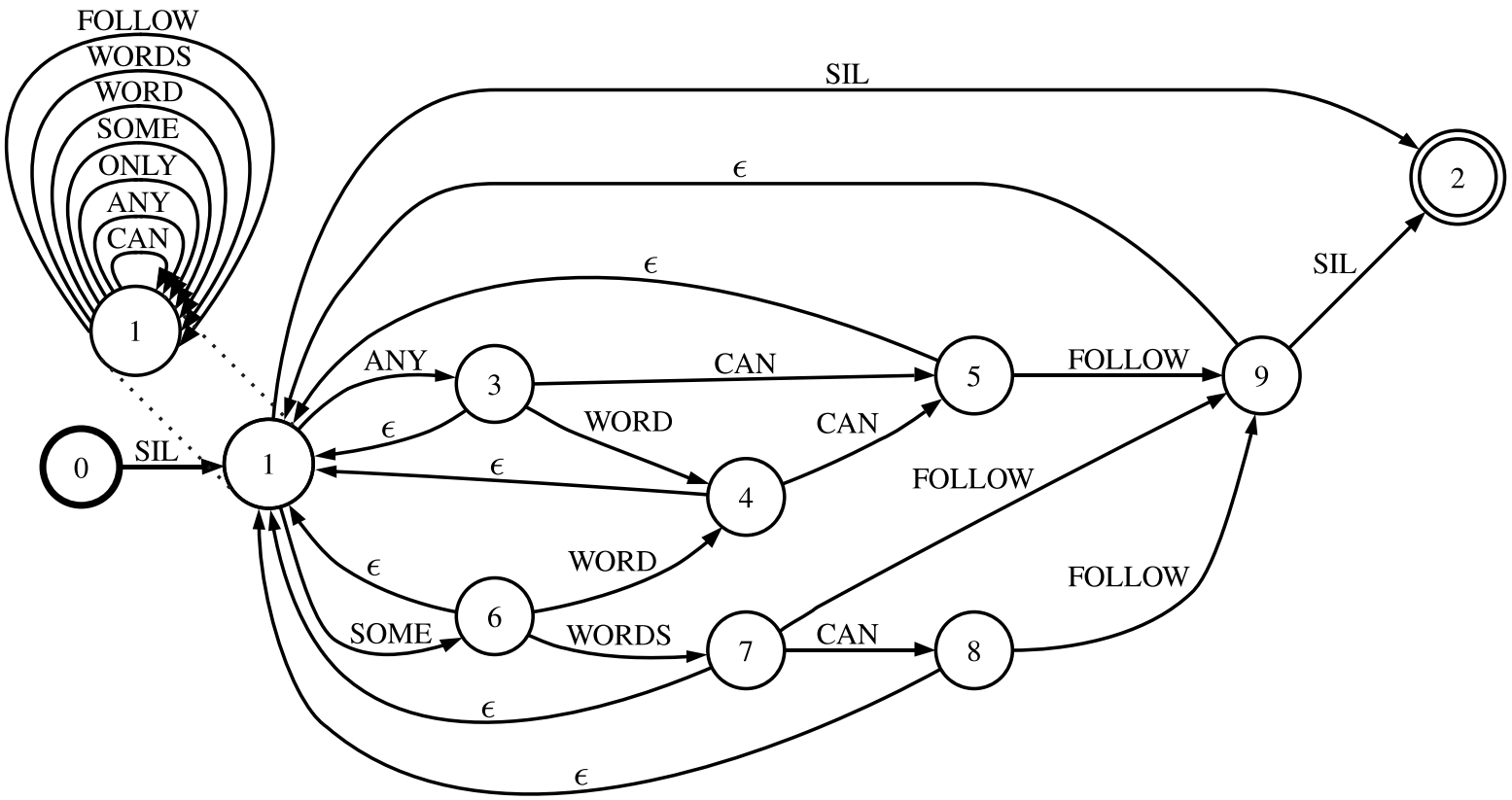


Figure: Statistical bigram. Node 1, the backoff node, is broken out for clarity.

Bigram Construction

- In a real bigram, the transitions leaving the bigram nodes would, in the simplest case, carry a weight determined from the bigram frequency statistics of a large training corpus of the form,

$$P(w_i|w_j) \approx \frac{N_{i|j}}{N_j},$$

  where $N_{i|j}$ is the number of times w_i was observed to follow w_j, and N_j is the total number of times w_j was observed in the context of any following word.
- As mentioned previously, the backoff node allows transitions labeled with any word in the vocabulary.

Bigram Construction (cont'd.)

- In a bigram, the weights on the transition from the backoff node labeled with word w_i are estimated from unigram word frequency counts of the form,

$$P(w_i) \approx \frac{N_i}{N},$$

  where N_i is the number of times w_i was observed in a training text in *any* context, and N is the total number of tokens in the training text.

- Transitions from bigram nodes to the backoff node are typically labeled with the null symbol ϵ, so that they can be taken without consuming any input. In practice, the transitions labeled with ϵ–symbols also carry weights determined by one of a number of backoff schemes.

Hybrid Language Models

- As both FSGs and statistical N–grams can be represented as WFSAs, hybrid structures with the features of both can also be constructed.
- In this way, it is possible to combine a well-structured task grammar with a statistical LM to provide flexibility when users stray outside the task grammar.

Pronunciation Dictionary

- The words of natural languages are composed of subword units in a particular sequence.
- In modern ASR systems, each word is typically represented as a concatenation of subword units based on its phonetic transcription.
- For example, the word “man” would be phonetically transcribed as “M AE N”, which is the representation in the recognition dictionary instead of the IPA symbols.
- The words of nearly any language can be covered by approximately 40 to 45 distinct phones.

Pronunciation Dictionary

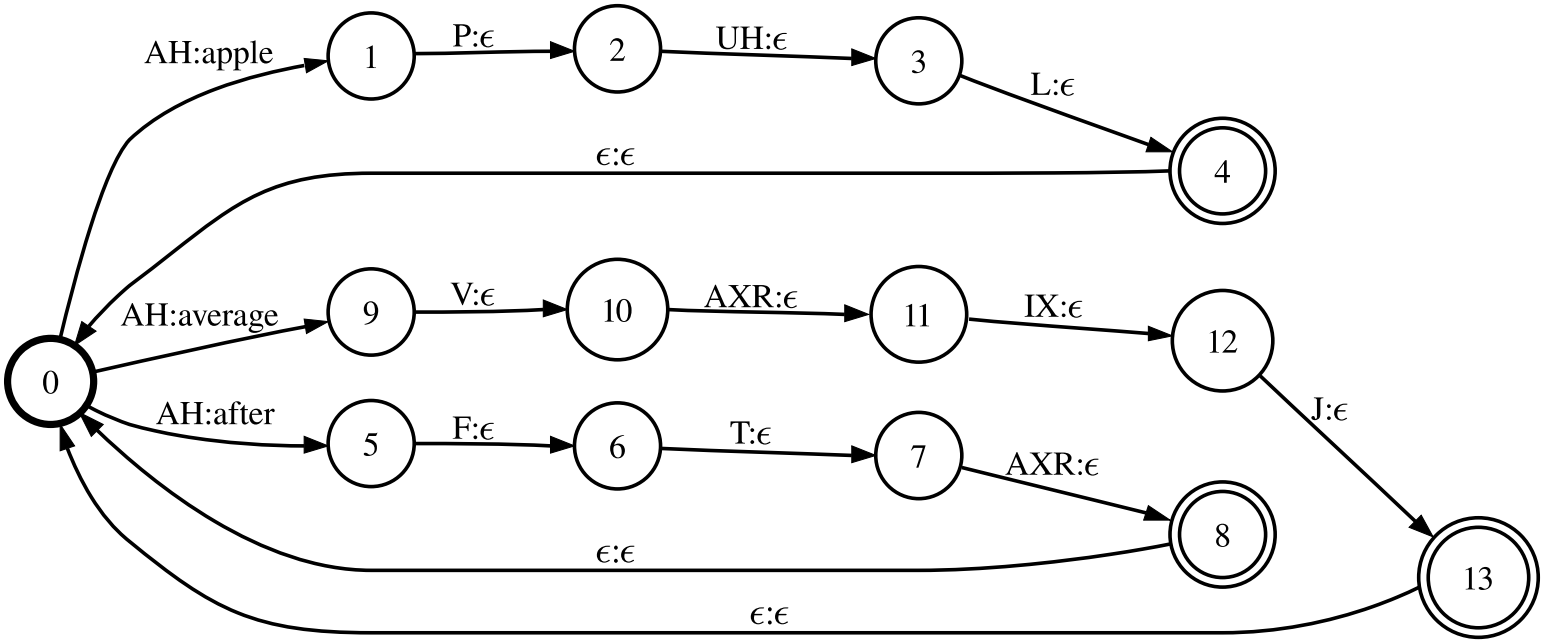


Figure: A pronunciation lexicon represented as a finite-state transducer.

Pronunciation Dictionary Schematic

- As shown in the figure, a pronunciation lexicon can be readily represented as a finite-state transducer L, wherein each word is encoded along a different branch.
- In order to allow for word sequences instead of only individual words, the ϵ–transitions from the last state of each word transcription back to the initial node are included.
- Typically L is *not* determinized, as this would delay the pairing of word symbols when L and G are composed.

Handling Homonyms

- Human languages like English contain many homonyms, like “read” and “red”.
- Such words have the *same* phonetic transcription, namely “R EH D”, but *different* spellings.
- Were the same phonetic transcription for both words simply added to a pronunciation transducer, and composed with a grammar G, the result would not be determinizeable, for the reasons given previously.
- As a simple remedy, auxiliary symbols such as #1 and #2 are typically introduced into the pronunciation transducer in order to disambiguiate the two phonetic, as shown in the following figure.

Handling Homonyms

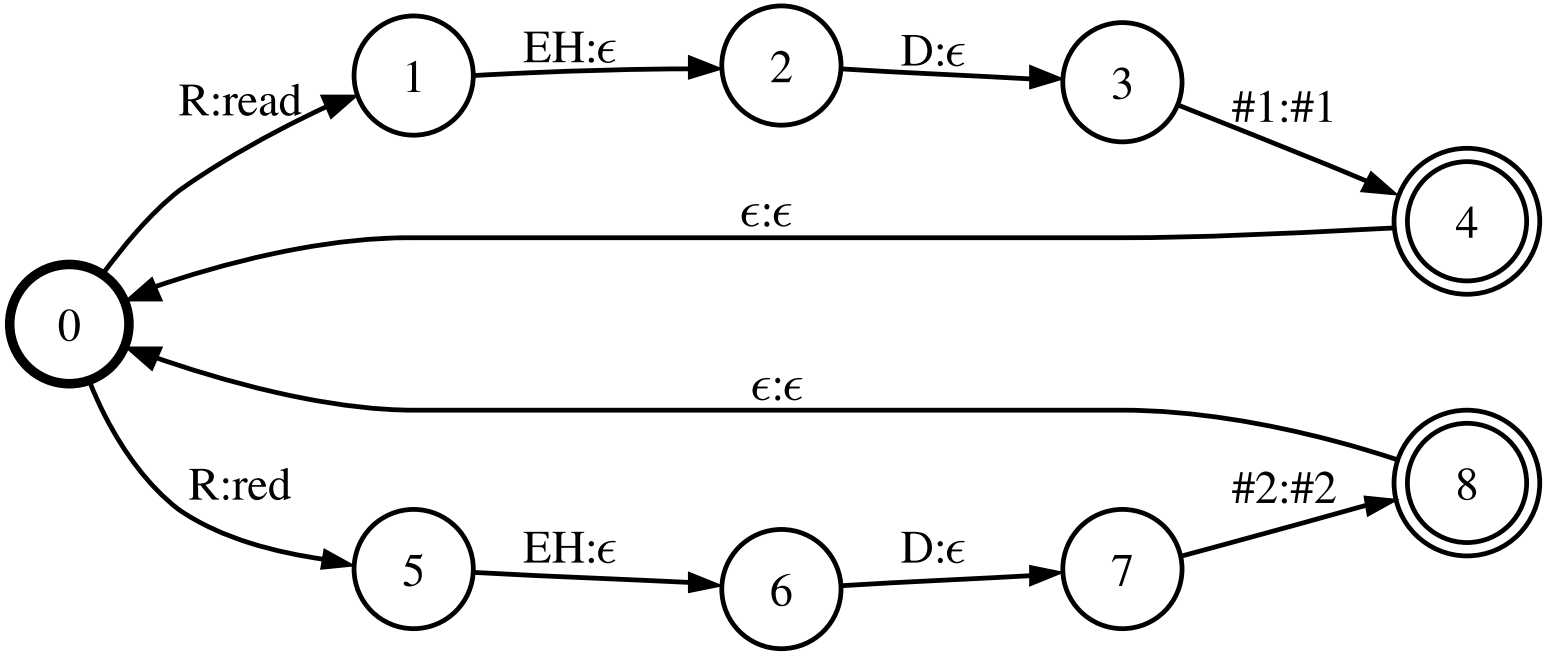


Figure: A pronunciation transducer with the auxiliary symbols #1 and #2.

Acoustic Model

- As previously mentioned acoustic representation of a word is constructed from a set of subword units known as phones.
- Each phone in turn is represented as a HMM, most often consisting of three states.
- The transducer H that expands each context independent phoneme into a three-state HMM is shown in the figure.
- In the figure, the input symbols such as “AH-b”, “AH-m”, and “AH-e” are the names of GMMs, which are used to evaluate the likelihoods of the acoustic features during the search process.
- The acoustic likelihoods are then combined with the LM weights appearing on the edges of the search graph in order to determine the shortest successful path through the graph for a given utterance, and therewith the most likely word sequence.

Acoustic Model Schematic

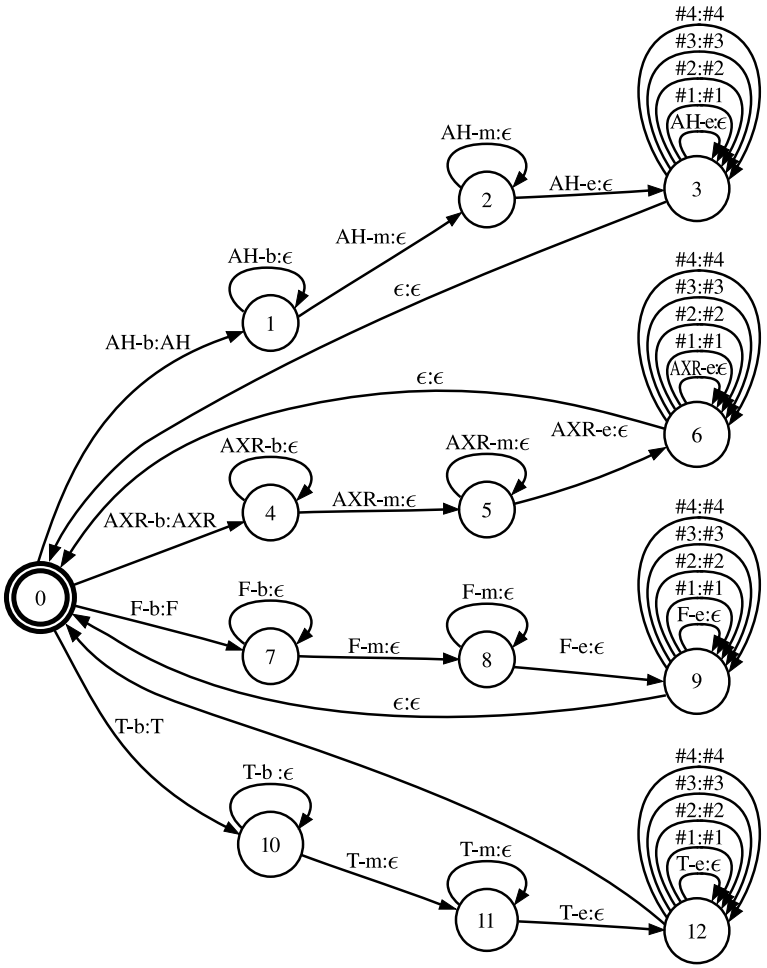


Figure: Transducer *H* specifying the HMM topology.

Modeling Coarticulation Effects

- As coarticulation effects are prevalent in all human speech, a phone must be modeled in its left and right context to achieve optimal recognition performance.
- A *triphone model* uses one phone to the left and one to the right as the context of a given phone. Similarly, a *pentaphone model* models considers two phones to the left and two to the right; a *septaphone model* models considers three phones to the left and three to the right.
- Using even a triphone model, however, requires the contexts to be clustered.
- This follows from the fact that if 45 phones are needed to phonetically transcribe all the words of a language, and if the HMM representing each context has three states, then there will be a total of $3 \times 45^3 = 273,375$ GMMs in the complete AM.

Modeling Coarticulation Effects (cont'd.)

- Such training could not be robustly accomplished with any reasonable amount of training data.
- Moreover, many of these contexts will never occur in any given training set for two reasons:
  1. It is common to use different pronunciation lexicons during training and test, primarily because the vocabularies required to cover the training and test sets are often different.
  2. State-of-the-art ASR systems typically use *crossword* contexts to model coarticulation effects between words.
- From the latter point it is clear that even if the training and test vocabularies are exactly the same, new contexts can be introduced during test if the same words appear in a *different order*.

Decision Tree

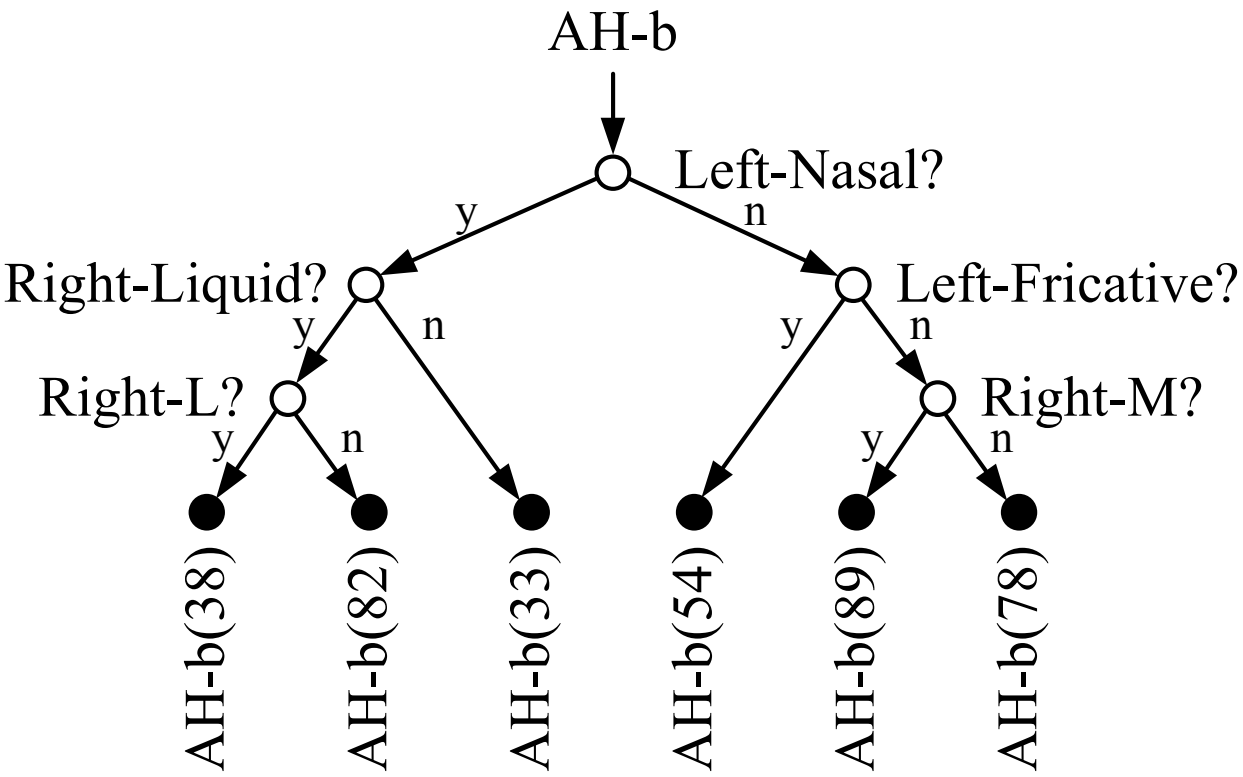


Figure: A decision tree for modeling context dependency

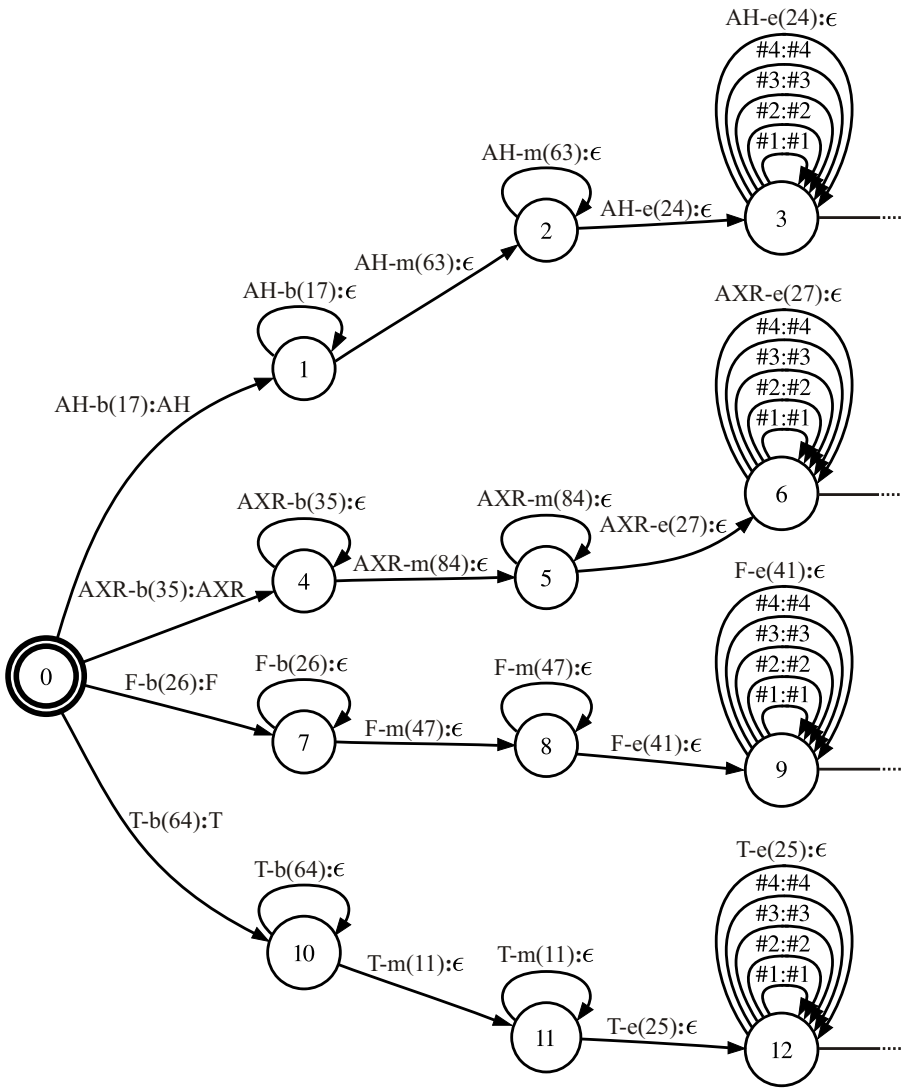
State Clustering

- A popular solution to these problems is to use triphone, pentaphone, or even septaphone contexts, but to use such context together with *context* or *state clustering*.
- With this technique, sets of contexts are grouped or clustered together, and all contexts in a given cluster share the same GMM parameters.
- The relevant context clusters are most often chosen with a *decision tree* such as that depicted in the next figure.
- As shown in the figure, each node in the decision tree is associated with a question about the phonetic context.
- The question “Left-Nasal” at the root node of the tree is to be interpreted as, “Is the left phone a nasal?”
- Those phonetic contexts for which this is true are sent to the left, the others to the right.

State Clustering (cont'd.)

- This process of posing questions and partitioning contexts based on the answer continues until a *leaf* node is reached, whereupon all contexts clustered to a given leaf are assigned the same set of GMM parameters.
- In order to model coarticulation effects during training and test, the context-independent transducer *H* is replaced with the context-dependent transducer *HC* shown in the following figure.
- The edges of *HC* are labeled on the input side with the GMM names (e.g., "AH-b(82)", "AH-m(32)", and "AH-e(43)") associated with the leaf nodes of a decision tree, such as that depicted in the prior figure.

Context-Dependency Transducer *HC*



Trellis Representation

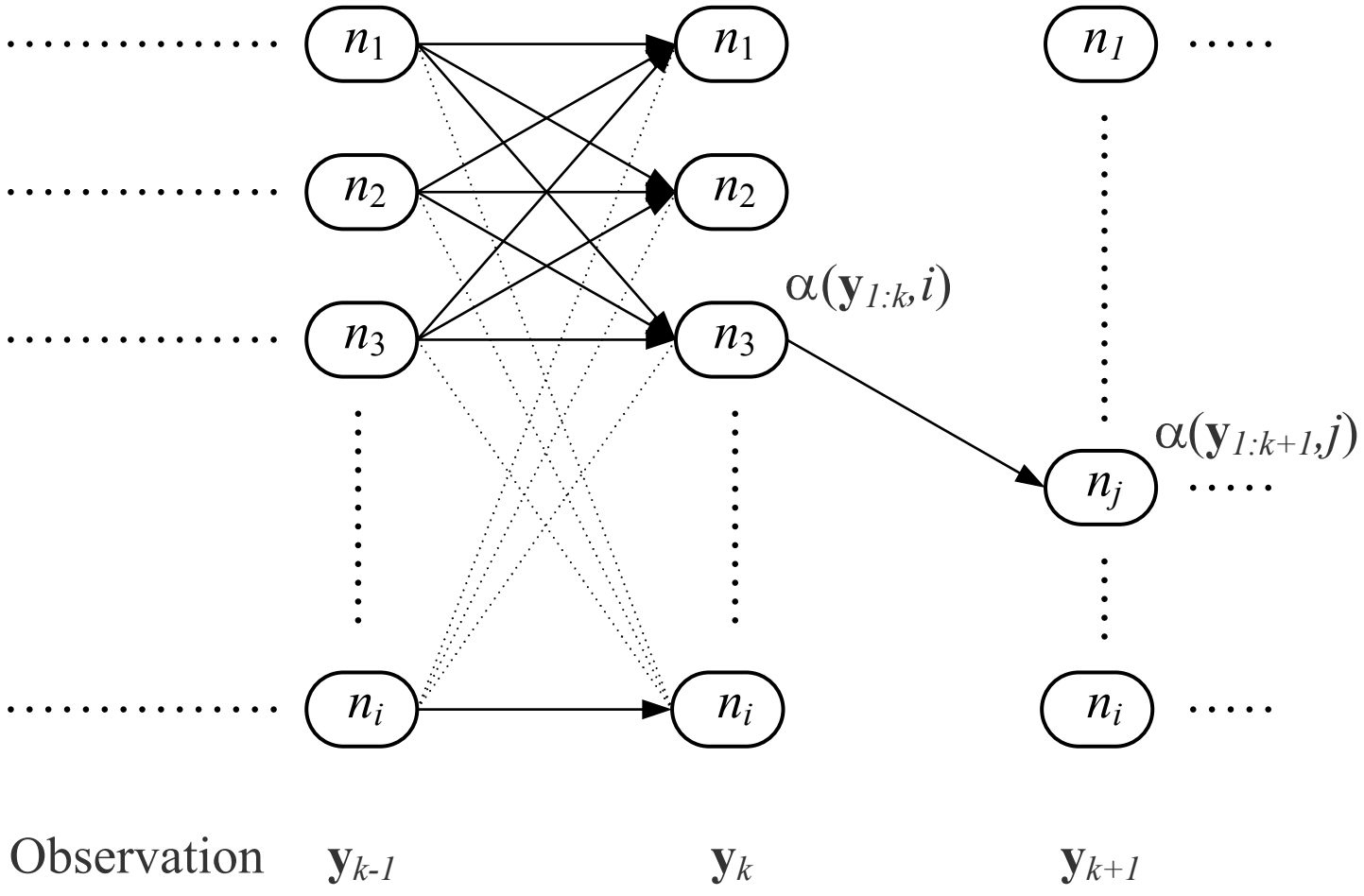


Figure: Trellis representation of Viterbi search.

Optimality of Viterbi Algorithm

- The nodes $v \in V$ of the trellis are specified by the pair $\{(q_i, k)\}$, where:
  - $q_i \in Q$ is a node in the original search graph, and
  - k is a time index.
- Thus, the trellis represents a *directed acyclic graph*.
- The Viterbi search is equivalent to relaxing the edges of the trellis in the order specified by a topological sort of the nodes.
- Hence, the Viterbi algorithm is correct; i.e., it returns the shortest path, which corresponds to the most likely hypothesis.

Context Free Grammars

There are four components of a context free grammar (CFG) $G = (V, T, P, S)$:

1. There is a set V of *variables* or *non-terminals*. Each variable represents a set of possible strings.
2. There is a set T of terminals and all strings w in L can be expressed as $w \in T^*$. Σ is the set of *terminals*.
3. There is a *start* symbol S.
4. There is a set P of *productions* that represent the recursive nature of the language.

Productions

Every production consists of:

1. A variable $A \in V$, the *head* of the production, which is partly defined through the production.
2. The production symbol $\rightarrow$.
3. the *tail* of the production, a string $\alpha \in (T \cup V)^*$ of one or more terminals and non-terminals.

CFG Example

Consider the context free grammar $G = (V, T, P, S)$ where

$V = \{$NP, NS, ART-SUB-M, VS, PP, PREP, PLACE, EXP-OBJ-M, PROJ-OBJ-M, V-PP, VS-H$\}$

$T = \{$Hund, Mann, Lastwagen, Typ, der, kommt, aus, nach, Badden-Baden, Berlin, Karlsruhe, Saarbrücken, Tübingen, den, gebissen, gesehen, gestreichelt, überfahren, hat$\}$,

$P = \{$

NP	$\rightarrow$	ART-SUB-M NS EXP-OBJ-M VS PP,
NS	$\rightarrow$	Hund | Mann | Lastwagen | Typ,
ART-SUB-M	$\rightarrow$	der,
VS	$\rightarrow$	kommt,
PP	$\rightarrow$	PREP PLACE | ϵ,
PREP	$\rightarrow$	aus | nach,
PLACE	$\rightarrow$	Baden-Baden | Berlin | Karlsruhe | Saarbrücken | Tübingen,
EXP-OBJ-M	$\rightarrow$	PRON-OBJ-M ART-SUB-M NS EXP-OBJ-M V-PP VS-H | ϵ,
PRON-OBJ-M	$\rightarrow$	den,
V-PP	$\rightarrow$	gebissen | gesehen | gestreichelt | überfahren,
VS-H	$\rightarrow$	hat $\}$.

Parse Tree

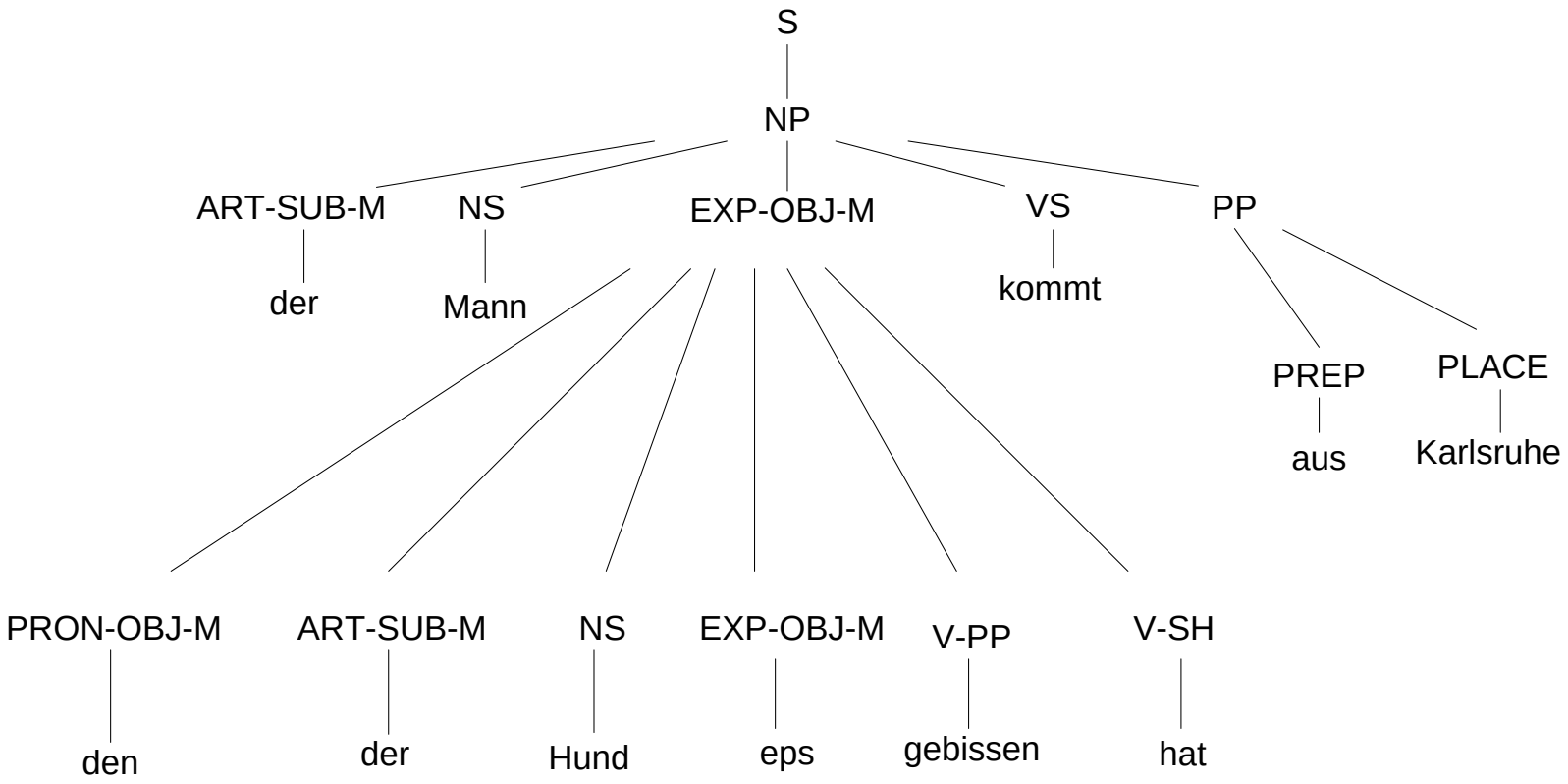


Figure: Parse tree G.

Finite-State Approximation

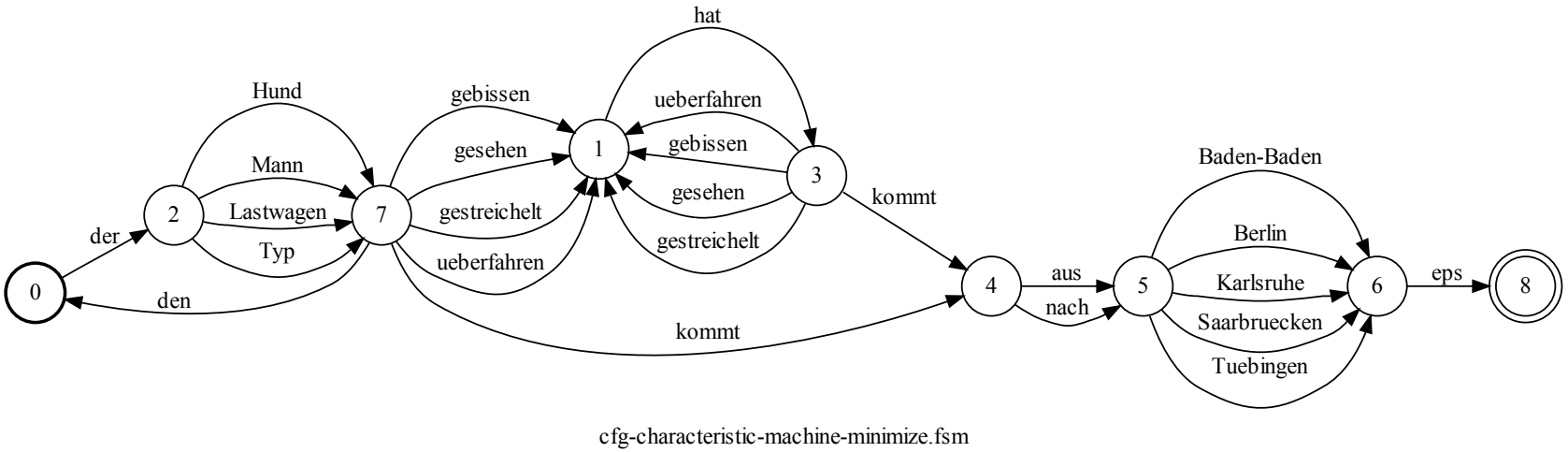


Figure: Finite-state approximation for the CFG *G*.

Summary

- In this tutorial, we discussed the application of weighted finite-state transducers in automatic speech recognition (ASR).
- We began with an examination of conventional finite-state automata, and the power set construction.
- Then we considered some conventional graph search algorithms.
- Next we defined weighted finite-state transducers (WFSTs), and generalized the power set construction to weighted determinization.
- We demonstrated how the knowledge sources used to build an ASR search graph can be represented as WFSTs.
- The Viterbi algorithm was presented as the search for the shortest path through a directed acyclic graph.
- Finally, we presented how context free grammars can be approximated with finite-state automata.