CS 4650/7650, Lecture 8
Semirings and Finite-State Transducers

Jacob Eisenstein

September 12, 2013

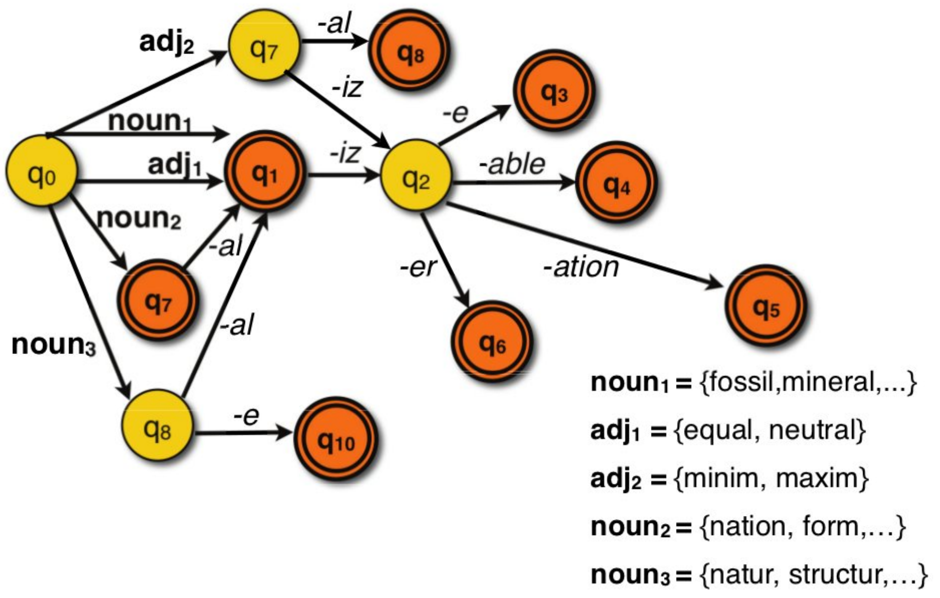
1 Homework and Project Review

1.1 Homework 3

- Did you figure out the Swahili example?
- What is the longest example of derivational morphology that you found?

2 Review of FSAs

- FSAs are constructions that decide if a string is in a regular language.
- WFSAs score the **acceptibility** of a string for a regular language.
- An FSA is a special case of a WFSA, where the only possible weights are $\{0, -\infty\}$.



3 Applications of WFSAs

We can use WFSAs to score derivational morphology as suggested above. But let's start with a simpler example:

Edit distance . We can build an edit distance machine for any word. Here's one way to do this (there are others):

- Charge 0 for "correct" symbols and rightward moves
- Charge 1 for self-transitions (insertions)
- Charge 1 for rightward epsilon transitions (deletions)

The minimum edit distance is the *sum* of costs across the best path through machine. Note that there may be other paths with higher costs.

Probabilistic models For probabilistic models, we make the path costs equal to the likelihood:

$$\delta(q_1, s, q_2) = P(s, q_2 | q_1) \tag{1}$$

The total score for a path is now the *product* of the transitions.
This enables probabilistic models, such as N-gram language models.

- A unigram language model is just one state, with V edges.
- A bigram language model will have V states, with V^2 edges.
- The reading [KM09] shows how to do an interpolated bigram/unigram language model. (Actually I think there's a smaller automaton, with only $V + 3$ states rather than $2V + 4$.)
  - Recall that the model is
    $$\hat{P}(y|x) = \lambda P_2(y|x) + (1 - \lambda)P_1(y), \tag{2}$$
    with $\hat{P}$ indicating the interpolated probability, P_2 indicating the bigram probability, and P_1 indicating the unigram probability.
  - Note that unlike the basic n-gram language models, our interpolated model has non-determinism: do we choose the bigram context or the unigram context?
  - For a sequence a,b, we want the final path score to be
    $$(\lambda P_2(a|\langle S\rangle)+(1-\lambda)P_1(a))(\lambda P_2(b|a)+(1-\lambda)P_1(a))(\lambda P_2(\langle E\rangle|a)+(1-\lambda)P(\langle E\rangle)) \tag{3}$$

Notice that a lot of the details are different between these three examples:

- Scoring
  - In the derivational morphology FSA, we wanted a boolean "score": is the input a valid word or not?
  - In the edit distance WFSA, we wanted a numerical (integer) score, with lower being better.
  - In the interpolated language model, we wanted a numerical (real) score, with higher being better.
- Nondeterminism
  - In the derivational morphology FSA, we accept if there is any path to a terminating state.

- In the edit distance WFSA, we want the score of the single best path.
- In the interpolated language model, we want to sum over non-deterministic choices.

- How can we combine all of these possibilities into a single formalism? The answer is semiring notation.

4 Semirings

A semiring is a system $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$

- $\mathbb{K}$ is the set of possible values, e.g. $\{\mathbb{R}_+ \cup \infty\}$, the non-negative reals union with infinity
- $\oplus$ is an addition operator
- $\otimes$ is a multiplication operator
- $\overline{0}$ is the additive identity
- $\overline{1}$ is the multiplicative identity

A semiring must meet the following requirements:

- $(a \oplus b) \oplus c = a \oplus (b \oplus c)$, $(\overline{0} \oplus a) = a$, $a \oplus b = b \oplus a$
- $(a \otimes b) \otimes c = a \otimes (b \otimes c)$, $a \otimes \overline{1} = \overline{1} \otimes a = a$
- $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$, $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$
- $a \otimes \overline{0} = 0 \otimes \overline{a} = \overline{0}$

Some semirings of interest:

Name	$\mathbb{K}$	$\oplus$	$\otimes$	$\overline{0}$	$\overline{1}$	Applications
Boolean	$\{0,1\}$	$\vee$	$\wedge$	0	1	identical to an unweighted FSA
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1	sum of probabilities of all paths
Log-probability	$\mathbb{R} \cup -\infty \cup \infty$	$\oplus_{\log}$	$+$	∞	0	negative log marginal probability
Tropical	$\mathbb{R} \cup -\infty \cup \infty$	min	$+$	∞	0	best single path

where $\oplus_{\log}(a, b)$ is defined as $-\log(e^{-a} + e^{-b})$.

Semirings allow us to compute a more general notion of the "shortest path" for a WFSA.

- Our initial score is $\overline{1}$
- When we take a step, we use $\otimes$ to combine the score for the step with the running total.
- When nondeterminism lets us take multiple possible steps, we combine their scores using $\oplus$.

Example Let's see how this works out for our language model example.

$$\begin{aligned} score(\{a,b,a\}) =& \overline{1} \\ & \otimes (\lambda P_2(a|*) \oplus (1-\lambda) \otimes P_1(a)) \\ & \otimes (\lambda P_2(b|a) \oplus (1-\lambda) \otimes P_1(b)) \\ & \otimes (\lambda P_2(a|b) \oplus (1-\lambda) \otimes P_1(a)) \end{aligned}$$

Now if we plug in the **probability semiring**, we get

$$\begin{aligned} score(\{a,b,a\}) =& 1 \\ & \times (\lambda P_2(a|*) + (1-\lambda) P_1(a)) \\ & \times (\lambda P_2(b|a) + (1-\lambda) P_1(b)) \\ & \times (\lambda P_2(a|b) + (1-\lambda) P_1(a)) \end{aligned}$$

- The score of the input will the **sum** of probabilities across all paths that successfully process the input.
- Note that if we really want to have a score that we can minimize, we should use the log-probability semiring, where the score will be the **negative** log-probability. Minimizing this is equivalent to maximizing the (log) probability.
- What happens if we set $\oplus = \max$?
- What semiring are we using for the edit distance machine?

5 Finite state transducers

FSAs and WFSAs apply to single strings. FSTs and WFSTs apply to pairs of string.

	acceptor	transducer
unweighted	FSA: $\Sigma^* \to \{0,1\}$	WFSA: $\Sigma^* \to \mathbb{R}$
weighted	FST: $\Sigma^* \to \Sigma^*$	WFST: $\Sigma^* \to \langle \Sigma^*, \mathbb{R} \rangle$

FSTs define **regular relations** over pairs of strings. We can think of them in a few different ways:

- **Recognizer**: accepts string pairs iff they are in the relation
- **Translator**: reads an input, produces an output

Formally, a finite-state transducer $M = \langle Q, \Sigma, \Delta, q_0, F, \delta, \sigma \rangle$ consists of:

- A finite set of states $Q = \{q_0, q_1, \ldots, q_n\}$
- Finite alphabets Σ for input symbols and Δ for output symbols
- Initial state $q_0 \in Q$ and final states $F \subseteq Q$
- A state transition function $\delta : \langle Q \times \Sigma^* \rangle \to 2^Q$
- A string transition function $\sigma : \langle Q \times \Sigma^* \rangle \to 2^{\Delta^*}$

Unlike NFSAs, not all NFSTs can be determinized. However, special subsets of NFSTs called **subsequential** transducers can be determinized efficiently (see 3.4.1 in [JM08]).

- A zeroth-order translation system could be made from a single state and a set of self-transitions: $Q_0 \xrightarrow[el]{the} Q_0$, $Q_0 \xrightarrow[los]{the} Q_0$, $Q_0 \xrightarrow[libro]{book} Q_0$, $Q_0 \xrightarrow[libros]{books} Q_0$,...
- First-order translation would require a state per word in the "input" vocabulary: $Q_0 \xrightarrow[\epsilon]{the} Q_{\text{the}} \xrightarrow[los\ libros]{books} Q_0$, $Q_{\text{the}} \xrightarrow[el\ libro]{book} Q_0$.
- Inflectional morphology and orthography:

$$Q_0 \xrightarrow[wit]{wit} Q_{\text{regular}} \xrightarrow[+s]{+\text{PL}} Q_1$$

$$Q_0 \xrightarrow[wish]{wish} Q_{\text{needs-e}} \xrightarrow[+es]{+\text{PL}} Q_1$$

6 Weighted FSTs

Weights can be added to FSTs in much the same way as they are added to FSAs.

- For any pair $\langle q \in Q, s \in \Sigma^* \rangle$, we have a set of possible transitions, $\langle q \in Q, t \in \Delta^*, \omega \in \mathbb{K} \rangle$, with a weight ω in the domain defined by the semiring.
- For example, we could augment the translation transducers defined above to allow alternative possible translations for a single word.
- The same semiring operations in WFSAs apply here too.

	acceptor	transducer
unweighted	FSA: $\Sigma^* \rightarrow \{0, 1\}$	WFSA: $\Sigma^* \rightarrow \mathbb{R}$
weighted	FST: $\Sigma^* \rightarrow \Sigma^*$	WFST: $\Sigma^* \rightarrow \langle \Sigma^*, \mathbb{R} \rangle$

Example : General edit distance computer.

- $Q_0 \xrightarrow[a]{a} Q_0 : 0$
- $Q_0 \xrightarrow[\epsilon]{a} Q_0 : 1$
- $Q_0 \xrightarrow[a]{\epsilon} Q_0 : 1$

6.1 Operations on FSTs

- Closed under **union**
- Closed under **inversion**, which switches input and output labels.
- Closed under **projection**, because FSAs are a special case of FSTs
- Not closed under **difference**, **complementation**, and **intersection**.
- Closed under **composition**.

FST composition is the basis for implementing the noisy channel model in FSTs, and can be used to support dozens of cool applications.

6.2 Finite state composition

Suppose we have a transducer T_1 from language I_1 to O_1, and another transducer T_2 from O_1 to O_2. The composition $T_1 \circ T_2$ is an FST from I_1 to O_2.

- Unweighted definition: iff $\langle x, z\rangle \in T_1$ and $\langle z, y\rangle \in T_2$, then $\langle x, y\rangle \in T_1 \circ T_2$.
- Weighted definition:

$$(T_1 \circ T_2)(x, y) = \bigoplus_{z \in \Sigma^*} T_1(x, z) \otimes T_2(z, y) \tag{4}$$

- Designing algorithms for composition is relatively straightforward if there are no epsilon transitions; otherwise it's more challenging [ARS09].

The simplest example

- $T_1 : Q_0 \xrightarrow[a]{x} Q_0, Q_0 \xrightarrow[b]{y} Q_0$
- $T_2 : Q_1 \xrightarrow{a} Q_1, Q_1 \xrightarrow{b} Q_2, Q_2 \xrightarrow{b} Q_2$
- $T_1 \circ T_2 : Q_1 \xrightarrow{x} Q_1, Q_1 \xrightarrow{y} Q_2, Q_2 \xrightarrow{y} Q_2$

For simplicity T_2 is written as a finite-state acceptor, not a transducer. Acceptors are a special case of transducers.

If we had weights, they would be combined through $\otimes$.

7 Applications of composition

7.1 Stemming

As discussed on Tuesday, information retrieval systems that only return exact matches aren't very useful.

- Suppose you query: *is it medically safe to kiss my cat on the lips*
- You want to get a hit even for documents like: *on the medical safety of kissing cats.*

- In morphologically complex languages like Hebrew, these differences could cause early IR systems to miss more than 90% of relevant documents [Cho89]!
- Stemming improves the **recall** of information retrieval systems by converting all tokens of *kissing* to *kiss*, etc.

The **Porter Stemmer** is a very popular stemming program. It is written as a set of rules, which are applied in stages, e.g.

- if the word ends in -*ing* and the preceeding part contains a vowel, delete the ending. (*hopping* $\rightarrow$ *hopp*)
- if the remaining part contains double letters (besides ss, ll, or zz), remove one (*hopp* $\rightarrow$ *hop*)

We can think of these rules as a sequence of deterministic finite state transducers:

- We can build a transducer to strip off endings like -*ing*, using two states to indicate whether we have yet seen a vowel
- Next we can build a transducer to strip off double letter endings, with exceptions for s, l, and z.
- We can **compose** these transducers into a single machine.

7.2 Back to edit distance

Remember the general-purpose edit distance FST? What happens if we compose that with a chain FSA for a word? We get the word-specific edit distance computer we talked about earlier.

- Composing an FST with a FSA yields a FSA.
- In many applications, we can build a **decoding** WFSA by composing a general-purpose WFST with an unweighted FSA representing the input.
- The best path through the resulting WFSA will be the minimum cost / maximum likelihood decoding.

7.3 Machine translation

- Simple models of machine translation can be implemented as finite-state transducers.

- Recall the example: *the books / los libros*. Here we are interested in modeling English-to-Spanish translation, $P(S|E)$.

  - Earlier we proposed a multi-state translation model to deal with the impact of pluralization on the Spanish article *los*
  - If we build our translator by composing a Spanish language model $P(S)$ with an Spanish-to-English transducer $P(E|S)$, this is not necessary. We can use the **noisy channel model**.
  - The $P(E|S)$ model can be a single state, as long as $P(S)$ models bigrams. A bigram model will tell us that $P(\textit{el libros}) \ll P(\textit{los libros})$.

    $$P(\textit{los libros}, \textit{the books}) = P_S(\textit{los}|\star) \otimes P_{E|S}(\textit{the}|\textit{los}) \otimes P_S(\textit{libros}|\textit{los}) \otimes P_{E|S}(\textit{books}|\textit{libros})$$

  - The composed FST can thus overcome its simplistic model of translation by having a more intelligent language model. This is useful, because language models can be trained without labeled data, while translation models cannot.
  - The general translation model looks like this: $T_S \circ T_{E|S}$. If we right-compose with an FSA representing the English string, we obtain a WFSA that scores all Spanish strings as $P(E,S) \propto P(S|E)$.
  - The reading [KM09] describes how we can add capabilities like adding and re-ordering words.

7.4 Morphological analysis

Recall that when we talked about morphology, there were several types of interacting rules:

1. To pluralize regular words, add an *s*; but if the word ends in *sh*, you have to add *es*.

2. To conjugate 3rd-person singular, add an *s*; but if the word ends in *sh*, you have to add *es*.

Pluralization and conjugation are different morphological systems, but once they have decided to append an s, the subsequent orthographical rules are the same.

This suggests an intermediate representation:

- *dish*+PL $\rightarrow$ *dish*$\wedge$*s*# $\rightarrow$ *dishes*
- *fish*+PL $\rightarrow$ *fish*# $\rightarrow$ *fish*
- *fish*+3S $\rightarrow$ *fish*$\wedge$*s*# $\rightarrow$ *fishes*
- Special symbols for morpheme and word boundaries
- The conjugation and pluralization FSTs only need to know what affix to add, and where to add it.
- Then an orthography FST takes over, and figures out how the morphemes should be combined.
- Finite-state composition allows us to automatically build a single machine for conjugation and pluralization, incorporating both the selection of affixes and orthographic constraints.
- Note that we have only described how to *generate* the English text. But if we compose this machine with a chain FSA representing an observed string, then we obtain an FSA where the set of accepted strings reveal the acceptable morphological analyses.
- For example, an utterance like *wishes* could have been produced by *wish/N*+3S or by *wish/V*+PL; both will be accepted by the resulting FSA.
- Suppose we want to distinguish the most likely morphological analysis given the context.
  - First, we add a loop back to the beginning in the morphological FST, so that we can accept multiple words.
  - Now we can build something like a language model WFST, but here we need probabilities of morphological analyses rather than just words. We might want to decompose this like

$$P(stem, affixes|context) \approx P(stem|context)P(affix|context) \tag{5}$$

  - If we have a morphologically annotated corpus, we can make smoothed relative frequency estimates of these probabilities. If we don't, what do we do?
  - The morphological analyses are missing data, so we might be able to do EM:
    * **E-step**: Decode all observed string sequences in the corpus, given the current probabilities.
    * **M-step**: Treat decoding as ground truth, update probabilities
    * Note: this is "Viterbi" EM, (a.k.a. "Hard" EM), because we are not computing probabilities over all possible decodings. We could do that using something called the expectation semiring [Eis01].

7.5 Context-sensitive spelling correction

Swype text entry in my old android phone does not consider context, much to my annoyance.

- I mean: *Prepare lecture for my class*
- It says: *Prepare lecture foie my class*

That's not smart. The bigram probabilities $P(\textit{lecture foie})$ and $P(\textit{foie my})$ should be ridiculously low (even dumber, it doesn't know that the unigram probability $P(\textit{foie}) \ll P(\textit{for})$, but that's another story...).

Rather than composing our edit distance computer with a single chain FSA, we can compose it with a language model FST — just like the language model FSA described above, but with output identical to the input.

The resulting machine is an FST with one state per word type.

- The cost of the transition from Q_a to Q_b on input s is the semiring product $\otimes$ of two costs:
  - The transition cost from a to b in the language model
  - The "emission" cost from s to a in the edit distance machine. Depending on exactly how we're getting our input, we could memorize these costs (for all pairs of words) and work at the token level,

rather than character level. But there's no conceptual difference; alternatively you could think about character-level edit distance FSTs sitting at each state.

- Given an input sequence s, we compose

  select the spelling-corrected string t with the greatest value,

  $$\hat{t} = \max_t \bigoplus_\pi s \rightsquigarrow_\pi t, \tag{6}$$

  where π is a path over input s and output t.

  - Why are there multiple paths from s to t? In the edit distance machine, we can always model matching input/output symbols as an insertion and a deletion.
  - Whether we care about these alternative paths depends on the semiring. In the tropical semiring, the solution is simply the minimum-cost path.

7.5.1 Norvig's spelling corrector

We haven't said much about where the weights come from. The bigram language model is probabilistic, and we can compute $\delta(q_s, q_t, t, t)$ as $-\log P(t|s)$. What about the edit distance model?

Before we get into that, here's an alternative approach, from Peter Norvig `http://norvig.com/spell-correct.html` (highly recommended). It may help explain how google is able to quickly and accurately spell-check your queries.

The approach can't easily be framed in exactly terms of the FST/semiring formalism, but it's close:

- Check if the word itself is in the dictionary. If so, return it.
- Else, check if any edit-distance=1 corrections is in the dictionary. If so, return the one with the highest unigram count.
- Else try corrections with edit-distance = 2.

The essay discusses extensions to bigrams a probabilistic model of errors. One way to build such a model is to look at data.

- Norvig suggests the Birbeck spelling error corpus, `http://norvig.com/spell-correct.html`.
  - Such a resource would tell us the likelihood of a word s being mispelled as t.
  - But new mispellings are always possible (even inevitable)
- An alternative would be to parametrize the edit distance model with more specific probabilities: P(insertion), P(deletion), etc.
  - Maximum-likelihood estimation would compute

$$P(\text{insertion}) = \frac{\text{count}(\text{insertion})}{\text{count}(\text{all-characters})} \tag{7}$$

  - But we will probably never get a corpus that gives us these counts.
  - Can we bootstrap our way to success? Suppose we could just guess the probabilities.
    * We could find the best path for each example in the training set, and keep track of the counts of insertions and deletions in these paths.

$$\begin{aligned}\hat{\pi}_{s,t} =& \arg\min_{\pi} \text{cost}(s \rightsquigarrow_{\pi} t)\\ \text{count}(\text{insertion}) =& \sum_{\langle s,t\rangle \in \mathcal{D}} \text{count}(\text{insertion}, \hat{\pi}_{s,t})\end{aligned}$$

    * More probabilistically, we could compute the likelihood of each path, and use these likelihood to find the *expected* counts:

$$\text{count}(\text{insertion}) = \sum_{\langle s,t\rangle \in \mathcal{D}} \sum_{\pi} \frac{P(s \rightsquigarrow_{\pi} t)}{\sum_{\pi'} P(s \rightsquigarrow_{\pi'} t)} \text{count}(\text{insertion}, \pi_{s,t})$$

    * Once we have the counts, we can go back and re-estimate the insertion probability (and all the other probabilities).
- This type of bootstrapping is another example of **expectation-maximization**.
  - We introduced EM in the simpler case of clustering.
    * Each document had a distribution over clusters $q(z)$

    * Each cluster had parameters θ.
    * The E-step computed $q(z)$, the M-step optimized θ.
  - Here, the Q distribution is over paths $Q(\pi)$.
  - In the E-step, we can compute $Q(\pi)$ given estimates of the parameters $P(\text{insertion})$.
  - In the M-step, we can update the parameters $P(\text{insertion})$ from expected counts under $Q(\pi)$.

- The number of paths can be very large, often too large to store.
  - In **Viterbi EM**, we just use the best path. This can work very well.
  - Alternatively, we can use the **expectation semiring** so that the "score" of the weighted path sum includes the expected counts [Eis01].

    Each edge is a tuple of a probability and a vector of expected counts, $\langle p_i, p_i v_i \rangle$

$$\langle p_i, p_i v_i \rangle \otimes \langle p_j, p_j v_j \rangle = \langle p_i p_j, p_j p_i v_i + p_i p_j v_j \rangle = \langle p_k, p_k v_k \rangle$$
$$\langle p_i, v_i \rangle \oplus \langle p_j, v_j \rangle = \langle p_i + p_j, v_i + v_j \rangle$$

  - If we can compute the semiring shortest path ($\mathcal{O}(n^3)$ at worst), we can compute the expected counts!

7.6 Speech Recognition

Speech recognition is yet another application of finite-state methods in NLP. It takes the idea of intermediate representations even further.

We want a mapping from intended words to observed acoustics. This can be seen as a multilevel process:

- G: WFST which generates English sentences: *I went to the bank*
- L: WFST which converts each word to context-indendent phonemes (WFST. Pronunciation dictionaries have this information): *Ay w eh n t t uw ...*
- C: WFST which converts context-independent phonemes to context-dependent phonemes: *Ay w eh n t uw ...*

- A: WFST which converts context-dependent phonemes to acoustic observations

By composing the FST $G \circ L \circ C \circ A$ with an FSA representing the observed acoustics O, we obtain a single WFSA which scores proposed English sentences for the observations O.

Inference The composed FSA would be ridiculously huge. Beam pruning is a technique for pruning away paths which are extremely unlikely, resulting in a faster, smaller FST.

Estimation It's really hard to get labeled data for all of these levels, for example context-dependent phones. We can treat this as a hidden variable and estimate it using EM.

Software There are mature software toolkits for working with finite state machines. OpenFST is a C++ package which I have had some experience with; it's fast and relatively well-documented. XFST and Carmel are other options.

References

[ARS09] Cyril Allauzen, Michael Riley, and Johan Schalkwyk. A generalized composition algorithm for weighted finite-state transducers. In *INTERSPEECH*, pages 1203–1206, 2009.

[Cho89] Yaacov Choueka. Responsa: A Full-Text retrieval system with linguistic processing for a 65-Million word corpus of jewish heritage in hebrew. *IEEE Data Eng. Bull.*, 12(4):22–31, 1989.

[Eis01] J. Eisner. Expectation semirings: Flexible em for learning finite-state transducers. In *Proceedings of the ESSLLI workshop on finite-state methods in NLP*, 2001.

[JM08] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Pearson Prentice Hall, 2 edition, May 2008.

[KM09] Kevin Knight and Jonathan May. Applications of weighted automata in natural language processing. In *Handbook of Weighted Automata*, pages 571–596. Springer, 2009.