Chapter 14:
Applications of Weighted Automata in Natural Language Processing

Kevin Knight and Jonathan May

USC Information Sciences Institute,
4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292, USA
knight@isi.edu
jonmay@isi.edu

1 Background

Linguistics and automata theory were at one time tightly knit. Very early on, finite-state processes were used by Markov [40, 30] to predict sequences of vowels and consonants in novels by Pushkin. Shannon [53] extended this idea

to predict letter sequences of English words using Markov processes. While many theorems about finite-state acceptors (FSAs) and finite-state transducers (FSTs) were proven in the 1950s, Chomsky argued that such devices were too simple to adequately describe natural language [6]. Chomsky employed context-free grammars (CFGs) and then introduced the more powerful *transformational grammars* (TGs), loosely defined in [7]. In attempting to formalize TG, automata theorists like Rounds [51] and Thatcher [57] introduced the theory of tree transducers. Computational linguistics also got going in earnest with Woods' use of *augmented transition networks* (ATNs) for automatic natural language parsing [59]. In the final paragraph of his 1973 tree automata survey [58], Thatcher wrote:

> The number one priority in the area [of tree automata] is a careful assessment of the significant problems concerning natural language and programming language semantics and translation. If such problems can be found and formulated, I am convinced that the approach informally surveyed here can provide a unifying framework within which to study them.

At this point, however, mainstream work in automata theory, linguistics, and computational linguistics drifted apart. Automata theorists pursued a number of theory-driven generalizations [15, 20, 21], while linguists went the other way and eschewed formalism. Computational linguistics focused for a time on extensions to CFGs [11, 52], many of which were Turing equivalent. In the 1970s, speech recognition researchers returned to capturing natural language grammar with FSAs, this time employing transition weights that could be trained on machine-readable text corpora [29, 28, 2]. These formal devices had associated algorithms that were efficient enough for practical computers of that time, and they were remarkably successful at distinguishing correct from incorrect speech transcriptions. In the 1990s, this combination of finite-state string formalisms and large training corpora became the dominant paradigm in speech and text processing; generic software toolkits for weighted finite-state acceptors and transducers (WFSAs and WFSTs) were developed to support a wide variety of applications [23, 46].

The twenty-first century has seen a reawakened interest in tree automata among computational linguists [34, 54, 25], particularly for problems like automatic language translation, where transformations are sensitive to syntactic structure. Generic tree automata toolkits [42] have also been developed to support investigations. In the remainder of this chapter, we discuss how natural language applications use both string and tree automata.

2 WFST Techniques for Natural Language Processing

In this section, we use two sample applications to highlight ways in which finite-state transducers are used in natural language processing. The first

application is transliteration of names and technical terms, and the second application is translation of natural language sentences. We conclude with a discussion of language modeling, an important part of both sample applications.

2.1 Example 1: Transliteration

Transliteration is the process by which names and technical terms are borrowed from one language to another. For some language pairs, this is a very simple or even trivial task—*Bill Gates* is written the same way in English and Spanish newspapers, and while the English word *conception* is changed to *concepción* in Spanish to preserve pronunciation; this is a regular and predictable pattern. However, the task becomes significantly more challenging when the language pairs employ different character sets and very different sound systems. For example, a Japanese newspaper may refer to アンジラナイト, using a sound-based character set called Katakana. If we know how Katakana encodes Japanese sounds, then we can sound out アンジラナイト as *anjiranaito.* The transformation from *anjiranaito* to some English term is still quite difficult, since among other constraints, Japanese words must end in vowels, and do not distinguish between l and r as is done in English.

After some thought, and perhaps the use of surrounding context, a bilingual speaker may realize *anjiranaito* was originally the English name *Angela Knight.* Here are more input/output samples:

マスターズトーナメント
`masutaazutoonamento`
`Masters Tournament`

アイスクリイーム
`aisukuriimu`
`Ice Cream`

ニューヨークタイムズ
`nyuuyookutaimuzu`
`New York Times`

Due to the large number of potential transliterations, this task is hard, even for humans. We can address the combinatorial explosion through the use of finite-state automata [33]. As a first attempt, we might contemplate a single finite-state transducer that converts a string of Katakana symbols K into strings of English letters from $\mathcal{E}$, the language of all English letter strings, with a corresponding probability of conversion for each English letter string E of $P(E|K)$, and chooses the most likely E in the language:

$$\underset{E \in \mathcal{E}}{\operatorname{argmax}} P(E|K) \tag{1}$$

The corresponding transducer design looks like this:

$$\text{Katakana} \Rightarrow \boxed{\text{WFST}} \Rightarrow \text{English}$$

This would be a very complex transducer to design. For example, the Japanese r sound may turn into the English letter R or the English letter L (or some other letter sequence), and this decision depends on many other decisions. We also want to guarantee that the English output phrase is well formed. $P(E|K)$ represents both of these probabilities in one complicated step. By using Bayes' law, we can separate the probability associated with well-formed E from the probability of transformation between K and E:

$$\operatorname*{argmax}_{E \in \mathcal{E}} P(E|K) = \operatorname*{argmax}_{E \in \mathcal{E}} \frac{P(E)P(K|E)}{P(K)} \tag{2}$$

$$= \operatorname*{argmax}_{E \in \mathcal{E}} P(E)P(K|E) \tag{3}$$

The corresponding transducer design now looks like this:

$$\boxed{\text{WFSA}} \Rightarrow \text{English} \Rightarrow \boxed{\text{WFST}} \Rightarrow \text{Katakana}$$

If we move from left to right, we can view this diagram as an explanation for Katakana strings. These explanations are often called "generative stories." According to this story, in order to produce a Katakana string, someone first generates a well-formed English phrase with probability $P(E)$ (according to the WFSA), and then someone converts that phrase into Katakana with probability $P(K|E)$ (according to the WFST). As generative stories go, this is actually a fairly realistic explanation of how Katakana words enter the Japanese vocabulary.

By contrast, if we move from right to left in the same diagram, we can convert a given Katakana string K into English by first sending it backward through the WFST, which will produce a multiplicity of English phrases that would transduce to K.[1] We can then intersect our multiplicity of phrases with the WFSA, in an effort to eliminate candidates that are not well formed.

This design, known as the *noisy-channel model* [53], has several advantages. First, the WFST can be greatly simplified, because it only models $P(K|E)$, the transformation of short English letter sequences into short Katakana sequences, and it does not need to pay attention to the global well formedness of the English. For example, an English T may nondeterministically transduce to a Katakana *to* or *ta*. The WFSA takes up the slack by enforcing global well formedness and assigns a $P(E)$ for any English string E, independent of any transformation. Also, we may have different resources for constructing the two devices—for example, we may have a large English dictionary to help us construct the WFSA.

---

[1] These English phrases can be represented as a finite-state acceptor, since the WFST preserves regularity in both directions.

The single WFST that represents $P(K|E)$ is still fairly complex. We would like to model the transformation in a series of small, easy-to understand steps. In this example, we break the initial transducer into a chain of three transducers, in the following design [33]:

$$\boxed{\text{WFSA A}} \Rightarrow \text{English} \Rightarrow \boxed{\text{WFST B}} \Rightarrow \text{English sounds}$$
$$\Rightarrow \boxed{\text{WFST C}} \Rightarrow \text{Japanese sounds} \Rightarrow \boxed{\text{WFST D}} \Rightarrow \text{Katakana}$$

According to this design, Katakana strings enter Japanese via the following path: (1) someone produces an English phrase, (2) that English phrase is converted into English sounds, (3) that English sound sequence is converted into Japanese sounds, and (4) those Japanese sounds are converted into Katakana symbols.

We justify this design in our probability model by using the conditional probability chain rule to break one probability distribution into a chain of independent distributions:

$$P(K|E) = \sum_{c \in C} P(K|c)P(c|E) \tag{4}$$

where C is any newly introduced parameter space that we can sum over.

This division can be repeated arbitrarily until we have the appropriate granularity of conditional probability, and hence WFST that we want for our model.

The probability model equation then becomes

$$\operatorname*{argmax}_{E \in \mathcal{E}} P(E|K)$$
$$= \operatorname*{argmax}_{E \in \mathcal{E}} \sum_{es} \sum_{js} \quad P(E) \cdot P(es|E) \tag{5}$$
$$\cdot P(js|es) \cdot P(K|js)$$

where es and js range over English and Japanese sound sequences, respectively.

Now that we have divided one complex automaton into a chain of automata, they are simple enough that we can build them—Fig. 1 shows fragments. WFSA A (Fig. 1a) nondeterministically generates English word sequences. WFST B (Fig. 1b) sounds out English word sequences. Note that this transducer can be used in either direction—given a sequence of words, forward application will output a sequence of sounds, and given a sequence of sounds, backward application will output a sequence of words. WFST C (Fig. 1c) converts English sounds into Japanese sounds. This is a highly nondeterministic process: an English consonant sound like T may produce a single Japanese sound t, or it may produce two Japanese sounds, such as $t\ o$ (as in the case of *Knight* transducing into *naito*). It is nondeterministic in the reverse direction as well, since a Japanese r sound may transduce to an English
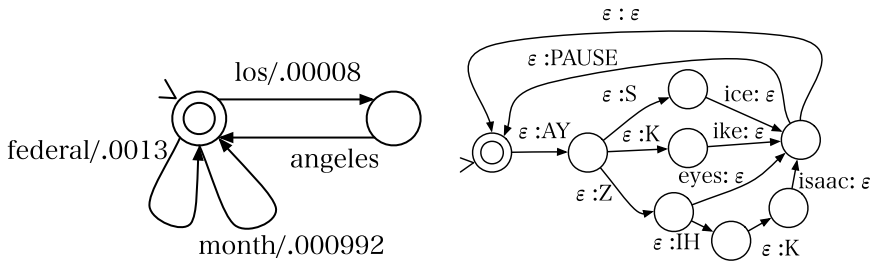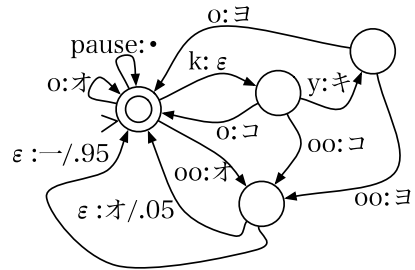
a WFSA A: Produce an English phrase.

b WFST B: Convert English phrase to English sounds.
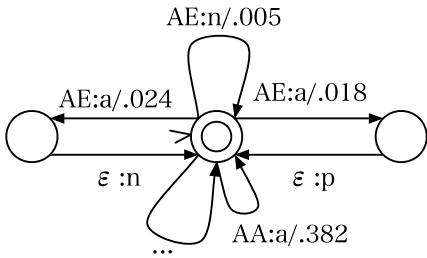
c WFST C: Convert English sounds to Japanese sounds.

d WFST D: Convert Japanese sounds to Katakana.

Fig. 1. Four automata fragments that transliterate English to Japanese [33]

R or L. However, the WFST can make its substitutions largely independent of context. Finally, WFST D (Fig. 1d) converts a Japanese sound sequence into Katakana writing. This is fairly deterministic, but requires some linguistic engineering to cover all the cases.

We can now translate a new Katakana string K by sending it backward through WFST D, then sending the result (which itself can be represented by a WFSA) backward through WFST C, and so on, finally intersecting it with WFSA A. In practice, this yields millions of English outputs, most of which consist of strange combinations of small (but legal) English words, e.g.,

```
Ann Gere Uh
Anne Jill Ahh
Angy Rugh
Ann Zillah
```
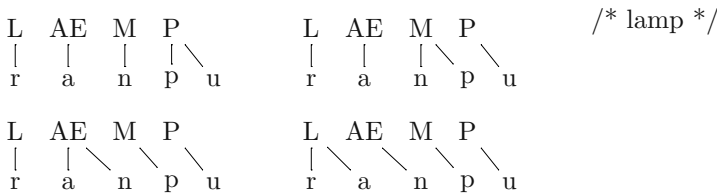
Here is where the probabilities from WFSA A and WFST C are important. If these are set to reflect what happens in the world (i.e., which English phrases are popular, and which sound substitutions are popular), then each potential

English output also comes with a score, and we can ask for the top-scoring ones (in order):

```
Angela Knight
Angela Nite
Andy Law Knight
Angela Nate
```

It turns out that passing our Katakana string through each transducer sequentially is only one of many possible search strategies. Another approach is to trivially transform A into a weighted transducer A' that is the partial weighted identity transducer with domain equal to the language accepted by A. We then compose transducers A', B, C, and D into a single weighted transducer, offline, then determinize and/or minimize it for deployment [45]; see Chap. 6. A third approach is to employ lazy composition [50, 63], which executes a parallel search through A', B, C, D, and K' (the identity transducer formed from the trivial automaton that accepts only K) by moving tokens from state to state in each. When the tokens all reach final states in their respective machines, an answer is generated; multiple answers can be created with backtracking or beam techniques.

What all of these strategies have in common is that they try to find the English word sequence(s) of highest probability, according to (5). Each of the factors in (5) is broken down further until we reach the probabilities actually stored on the transitions of our WFSAs and WFSTs. For example, consider $P(js|es)$. In Fig. 1c, we can see that WFST C converts English sounds into Japanese sounds via a one-to-many substitution process. Given an English sound sequence es and a Japanese sound sequence js, our WFST can convert es to js in several different ways, depending on how the individual English sounds take responsibility for subsequences of js. Each way can be represented by an alignment that specifies, for each Japanese sound, which English sound produced it. For example, there are four ways to align the sound sequences (L AE M P, r a n p u):

```
L   AE   M   P              L   AE   M   P         /* lamp */
|    |   |   | \            |    |   |  \   \
r    a   n   p   u          r    a   n   p   u

L   AE   M   P              L   AE   M   P
|    |  \   \   \           |  \   \   \   \
r    a   n   p   u          r    a   n   p   u
```

Each alignment corresponds to a different transducing path through WFST C. While we may prefer the first alignment, the others may exist with some small probability. We therefore write the total probability of js given es as:

$$P(js|es) = \sum_a \prod_{i=1}^{|es|} P(j_{\mathrm{seq}_{es_i}}|es_i) \tag{6}$$

where alignment a maps each English sound es_i onto Japanese sound subsequence $j_{\text{seq}_{es_i}}$.

Where do transition probability values come from? It is hard for human designers to produce these numbers, so we typically learn them from online text corpora. In the case of WFSA A, we may gather English word frequencies and normalize them. For example, if we see the word *the* 1,200 times in a corpus of 12,000 words, we can assign $P(the) = 0.1$. In the case of WFST C, we can collect probabilities from manually aligned sequence pairs. For example, notice that the English sound M occurs twice in the following database:

```
L   AE  M   P                S   T   IY  M
|   |   |   | \              | \   \    \ \ \
r   a   n   p  u             s   u   t   i  i  m  u
```
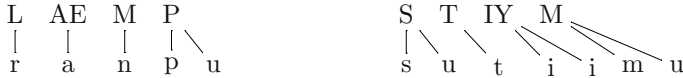
From this, we may conclude that $P(n|M) = 0.5$ and $P(mu|M) = 0.5$. Of course, it is important to have thousands of such pairs, in order to get accurate probability estimates.

For both WFSA A and WFST C, what justifies this "count and divide" strategy for estimating probabilities? Here, we have followed the *maximum likelihood* principle [18], assigning those scores to our transitions that maximize the probability of the training corpus. This principle is especially handy when our training corpus is incomplete. For example, we may only have access to a plain (unaligned) bilingual dictionary:

```
L   AE  M   P                S   T   IY  M
r   a   n   p   u             s   u   t   i   i   m   u
```

and many other pairs.

Given any particular set of parameter values, such as $P(n|M) = 0.32$ (and so on), we can compute $P(js|es)$ for each example pair and multiply these together to get a corpus probability. Some sets of values will yield a high corpus probability, and others a low one. The *expectation–maximization* (EM) algorithm [12] can be used to search efficiently for a good set of values. In this case, highly accurate alignments can be generated automatically without human intervention. Other popular methods of parameter estimation include maximum entropy [27] and minimum error-rate [13].

2.2 Example 2: Translation

We now turn to our second sample application, automatic translation of sentences. This is more challenging for several reasons:

- There are hundreds of thousands of distinct words, versus dozens of distinct linguistic sounds.
- Each word may have many context-dependent meanings or translations.

- Translation often involves significant reordering. For example, in English, the verb comes in the middle of the sentence, while in Japanese, it comes at the end.
- Ensuring that our output is globally well formed requires capturing vast amounts of knowledge about the syntax of the target language, in addition to semantic understanding of how the world works.

While the automatic translation problem remains unsolved, substantial progress has been made in recent years. Much of this progress is due to automatic analysis of large manually-translated documents, such as are produced each year by the United Nations and the European Union.

We might start with the following design for translation:

$$\text{Spanish} \Rightarrow \boxed{\text{WFST}} \Rightarrow \text{English}$$

This design is again problematic, because each word must be translated in the context of all the other words. Therefore, we employ the noisy-channel model approach from Sect. 2.1:

$$\boxed{\text{WFSA}} \Rightarrow \text{English} \Rightarrow \boxed{\text{WFST}} \Rightarrow \text{Spanish}$$

In this scheme, the WFST can operate in a largely context-independent fashion. Sending a particular Spanish sentence backward through the WFST might yield many target hypotheses, e.g.,

```
John is in the table
John is on the table
John on is the table
       etc.
```

When we intersect this set with the English WFSA, grammatical hypotheses can be rewarded. Note that the WFSA helps out with both word choice and word ordering, and we can train this WFSA on vast amounts of monolingual English text.

How about the WFST? Brown et al. [4] proposed a particular model for $P(s|e)$ which would assign a conditional probability to any pair of Spanish and English strings. Knight and Al-Onaizan [32] cast this model as a sequence of finite-state automata. Figure 2 depicts the operation of these automata, and Fig. 3 shows automata fragments.

WFSA A (Fig. 3a) generates English word sequences according to some probability distribution that (we hope) assigns high probability to grammatical, sensible sequences. WFST B (Fig. 3b) decides, for each English word, whether to drop it, copy it, duplicate it, triplicate it, etc. The decision is
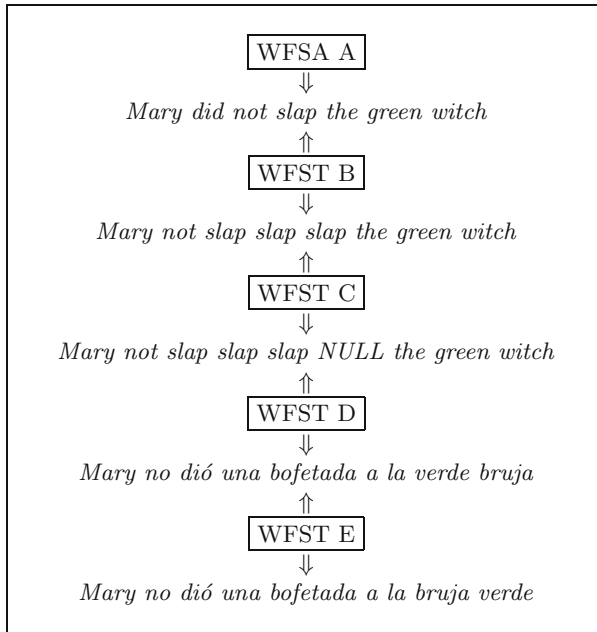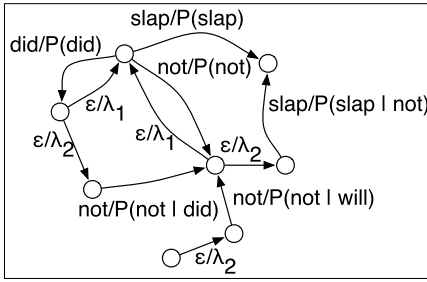
WFSA A
⇓
Mary did not slap the green witch
⇑
WFST B
⇓
Mary not slap slap slap the green witch
⇑
WFST C
⇓
Mary not slap slap slap NULL the green witch
⇑
WFST D
⇓
Mary no dió una bofetada a la verde bruja
⇑
WFST E
⇓
Mary no dió una bofetada a la bruja verde

Fig. 2. The generative model of [4] as a cascade of automata

based only on the word itself, with no context information. After each result-
ing word, WFST C (Fig. 3c) inserts a NULL token[2] with probability 0.02.
WFST D (Fig. 3d) then translates each word, one for one, into Spanish. Fi-
nally, WFST E (Fig. 3e) reorders the resulting Spanish words. Each transducer
is simple enough to build; all of them are highly nondeterministic.

We do not use this transducer cascade in the forward direction, but rather
in the reverse direction, to translate Spanish into English. We begin by sending
our Spanish input backward through WFST E, to obtain various reorderings,
including what we hope will be an English-like ordering. Ultimately, the results
are intersected with WFSA A, which is designed to prefer well-formed English.
Because of the scale of this problem, translating like this requires pruning the
intermediate results. However, it is likely that we will accidentally prune out
a good hypothesis before the WFSA A has had a chance to reward it. In
practice, therefore, we must perform an integrated search in which all the
automata weigh in simultaneously during the incremental construction of the
English translation.

How are translation probabilities estimated? We first obtain quantities of
manually-translated documents and process them into sentence pairs that are
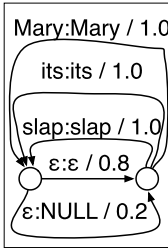mutual translations. If we were provided with word alignments, e.g.,

---

[2] The NULL word is designed to generate Spanish function words that have no
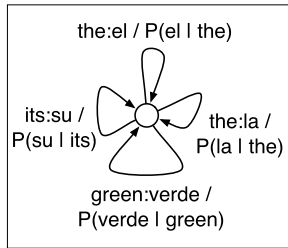English equivalent.
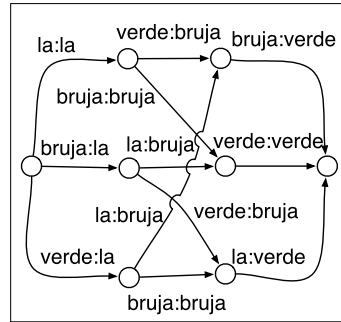
a WFSA A: Produce a phrase.
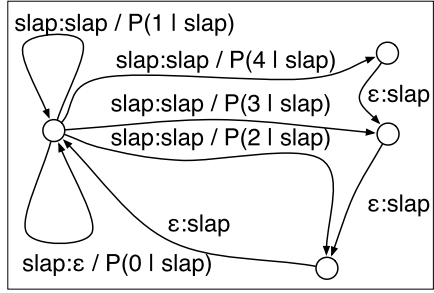
b WFST B: Delete or copy words.

c WFST C: Insert NULL tokens.

d WFST D: Translate words.

e WFST E: Re-order words.

Fig. 3. Five automata fragments that translate Spanish to English. WFSA A produces an English phrase, and WFSTs B–E transform that phrase into Spanish

then we could estimate the parameters of WFSTs B, C, D, and E. For example, out of 1,000 alignment links connected to the word "green," perhaps 250 link to "verde," in which case $P(verde|green) = 0.25$. However, we are never provided with such alignment links, so again, we use the EM algorithm to guess both links and probability values.

The particular translation model of Fig. 3 was one of the first to be designed, and empirical experiments have revealed many weaknesses. One is that it translates word to word, instead of phrase to phrase. While it has the
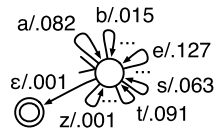
Fig. 4. WFSA for a 1-gram letter language model

capability for phrasal translation (as in *slap ⇔ dio una bofetada*), it does not execute such substitutions in a single step. More recent models, taking advantage of more computational power, remedy this. Other problems are more serious. For example, it is difficult to carry out large-scale reordering with finite-state machines, and it is difficult to make these reorderings sensitive to syntactic structure—e.g., the verb in English must somehow move to the end of the sentence when we translate Japanese. Furthermore, it is difficult to attain globally correct outputs, since the well formedness of English depends to some extent on hierarchical, nesting structure of syntactic constituents. For these reasons, some recent models of translation are appropriate for casting in terms of tree automata rather than string automata [60, 1, 61, 22, 16, 43, 19], and we investigate such models later in this chapter.

2.3 Language Modeling

In the previous sections, we focused on the transducers specific to each application. Here, we focus on *language modeling*, the problem of appropriately representing $P(E)$, a WFSA that models well-formed English sentences. Language models are used in any natural language application concerned with well-formed final output.

Shannon [53] observed that the generation of natural language text could be approximated to a reasonable measure by a WFSA that uses states to encode recently seen context. A simple example is a *1-gram* language model of characters, which simply encodes individual character frequencies. If in a corpus of 1,000,000 English characters, the letter e occurs 127,000 times, we estimate the probability $P(e)$ as 127,000/1,000,000, or 0.127. This model can be represented as a WFSA, as shown in Fig. 4.

A 2-gram model remembers the previous letter context—its WFSA has a state for each letter in the vocabulary. The transition between state r and state e outputs the letter e and has probability $P(e|r)$. We can train n-gram models in this way for any n. If we use such models to stochastically generate letter sequences, we observe the following results:

1-gram: thdo cetusar ii c ibt deg irn toihytrsen ...
2-gram: rt wo s acinth gallann prof burgaca ...
3-gram: restiche elp numarin cons dies rem ...
4-gram: what the legal troduce inortemphase ...
5-gram: we has decide in secuadoption on a ...
6-gram: thern is able to the bosnia around ...

While the 6-gram model generates more word-like items than the 1-gram model does, it still lacks sufficient knowledge of English grammar. For noisy-channel applications like translation and speech, a language model needs to know much more, in order to make decisions involving word choice and word order. Work in speech recognition in the 1970s and 1980s effectively trained and used *word* n-gram models, where the probability of a word depends on the previous $n-1$ words; since then, word n-gram models have been the dominant form of language model used in practical systems. This is somewhat surprising, given the work of Chomsky in the 1950s and 1960s which claimed that finite-state string processes were unsuitable for representing human grammars [6, 7, 44]. The largest language model built to date is a 7-gram model, built from one trillion words of English [3] and used for automatic language translation.

A language model should not assign zero probability to any string. For example, a 3-gram language model should accept a string even if it contains a word-triple that was never observed before in training. The process of *smoothing* reassigns some of the probability from seen events to unseen events. One simple technique is *interpolation* smoothing. For the 2-gram case, where we are calculating the likelihood of seeing word y given that the last recognized word was x, instead of estimating $P(y|x)$ as $\frac{\text{count}(xy)}{\text{count}(x)}$, which might be zero, we interpolate with the 1-gram probability of y:

$$P(y|x) = \lambda_1 \cdot \frac{\text{count}(xy)}{\text{count}(x)} + (1 - \lambda_1) \cdot \frac{\text{count}(y)}{N} \tag{7}$$

where N is the size of the training corpus. Likewise, $P(z|x,y)$ can be estimated as $\lambda_2 \frac{\text{count}(xyz)}{\text{count}(xy)} + (1 - \lambda_2)P(z|y)$. Once the counts have been collected from a training corpus, the λ_i values can be set to maximize the likelihood of a smaller (held-out) smoothing corpus, via the EM algorithm. Language models are often evaluated on the probability P they assign to a (further held-out) blind test corpus, or on the *perplexity*, which is $2^{\frac{-\log(P)}{N}}$.

Interpolation smoothing is not the best smoothing method available, but it can be implemented directly in a WFSA, as shown in Fig. 5. This formulation is space efficient, requiring only one transition per observed n-gram, rather than one transition per conceivable n-gram.

3 Applications of Weighted String Automata

In Sect. 2, we saw details of how WFSAs and WFSTs can be used to implement noisy channel models for two applications. In this section, we review recent work in other areas of natural language processing that uses similar techniques. In most cases, the structures and designs, though described in varied ways, are very similar and only differ in the data being modeled.
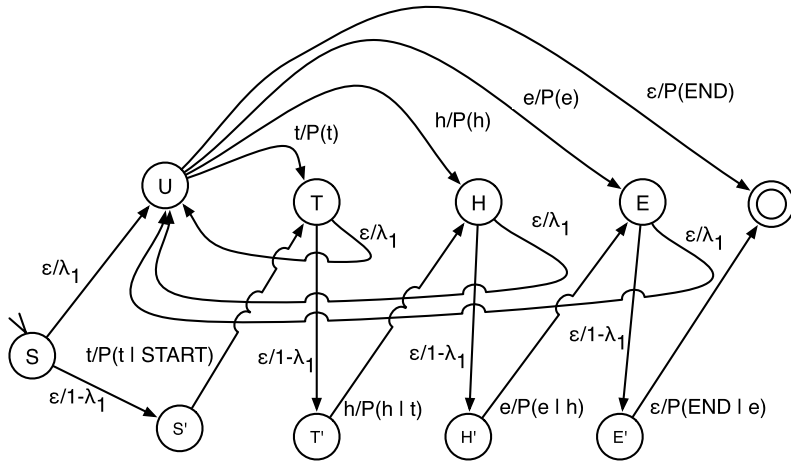
Fig. 5. Fragment of a WFSA for a 2-gram letter language model. At each state {S, T, H, E}, a decision is made to use 2-gram context by moving to states {S', T', H', E'}, respectively, or to use 1-gram context by moving to state U

3.1 Language Translation

We described a word-for-word model of language translation in Sect. 2. This model was implemented in a WFST framework by [32]. A phrase-for-phrase model was subsequently devised by [47] and implemented in a WFST framework by [37]. Translations from this model are much more accurate, and by using a WFST toolkit, Byrne et al. [37] are able to build a cascade of transducers and execute translations using generic finite-state procedures. The most problematic transducer is the one responsible for reordering—such a general transducer would be exceedingly large if built offline. In practice, given a particular source-language sentence, we can encode it and all of its local reorderings online as a temporary WFSA, which is then sent through the rest of the noisy-channel cascade.

3.2 Speech Recognition

Pereira et al. [49] apply the noisy-channel framework to the problem of speech recognition, i.e., recovering the sequence of spoken words that generated a given acoustic speech signal. A standard n-gram language model like that described in Sect. 2.3 is used. The noisy channel transducer, which generates $P(E|S)$ for a received acoustic speech signal S, is described as a chain of transducers as follows:

- For each word in S, a variety of phone sequences, i.e., individual units of speech, may be observed that can be interpreted as the word, with varying probabilities. For each word, a word-to-phone transducer is constructed,

and the closure of these transducers over all words forms the complete word-to-phone transducer.

• Similar to the word-to-phone model, each phone can be expressed as a variety of audio signals. Again, the closure of phone-to-audio transducers for each phone is taken as the complete phone-to-audio transducer.

Once defined, the chain of transducers and the final language model are weighted with the method of maximum likelihood, directly observing probabilities from available training data, and possibly smoothing. Composition and decoding are handled entirely by generic automata operations as, for example, implemented in the AT&T FSM Toolkit [46].

3.3 Lexical Processing

In most natural language applications, it is necessary to cut an information stream into word units. This is especially hard in languages without whitespace, such as Chinese. Sproat et al. [56] show how to automatically break Chinese into words by constructing a series of WFSTs. Word-internal units must also be processed. We saw this in the case of transliteration (Sect. 2.1). Another problem is morphological analysis, in which a word is analyzed into *morphemes*, the smallest units of language that carry meaning. Languages like Turkish and Finnish are written with very long words that must often be broken into what would be equivalently represented by separate articles, prepositions, and nouns in other languages. For many other languages, simply finding the root form of an inflected word is a challenge. One of the most successful early introductions of finite-state processing into natural language processing was for morphology [31], and a weighted approach can be found in [9].

3.4 Tagging

A wide variety of natural language problems can be cast as *tagging* problems, in which each word of input is assigned a tag from some finite set. The classic example is part-of-speech tagging, which seeks to disambiguate the syntactic category of each word in a sentence. Given the sentence *The flag waves in the wind*, the tagger must realize that *flag* and *wind* are nouns, even though both can be verbs in other contexts (e.g., *wind a watch*). Finite-state methods are often applied to this task [8]; within the noisy channel framework, we can build an n-gram WFSA to model grammatical tag sequences, and a one-state WFST to model substitutions of words by tags. Another common tagging problem is to locate named entities (such as people, places, and organizations) in texts. Here, each word is tagged as either B (word begins a new entity), I (word is inside an entity), or O (word is outside an entity). This ternary tagging scheme covers cases where two entities may be adjacent in text. A sequence like *Japan gave Russia the Kuriles* would be tagged B O B B I.

3.5 Summarization

Text summarization is the shrinking of a document or set of documents into a short summary that contains a useful subset of the information. One application of summarization, headline generation, drops unnecessary words from an input text and performs limited transformation of the remaining words to form an appropriate news headline. The noisy-channel framework is followed to accomplish this task in [62], where the source is considered to be emitting a series of compressed sentences in "headlinese" which are then passed through a transducer that inserts extra words and transforms some words to form grammatical expanded sentences. Zajic et al. [62] tweak their results by introducing various penalties and feature weights onto the transducer arcs; these can be modeled by modifying weights accordingly or by introducing additional transducers that explicitly encode the transitions.

3.6 Optical Character Recognition

The automatic conversion of hard-copy printed material to electronic form is useful for preserving documents created before the digital age, as well as for digitizing writing that is still generated in a nondigital manner, e.g., converting handwritten notes. Scanner technology has progressed considerably in recent years thanks to probabilistic recognition techniques, which are representable in the noisy channel framework. Here, the noise metaphor is readily apparent; clear, uniformly represented characters are garbled by the noisiness of the printed page, incorrectly struck typewriter keys, or the human hand's inconsistency. Kolak et al. [36] use this approach, and built their final system with the AT&T FSM toolkit [46], thus using automata operations directly. The chain of transducers in this case first segments the words into characters, then groups the characters into subword sequences, and finally transforms the sequences into noise-filled sequences.

4 Applications of Weighted Tree Automata

String WFSTs are a good fit for natural language problems that can be characterized by left-to-right substitution. However, their expressiveness breaks down for more complex problems, such as language translation, where there is significant reordering of symbols, and where operations are sensitive to syntactic structure.

The usefulness of hierarchical tree structure was noticed early in linguistics, and automata theorists devised tree acceptors and transducers [51, 57] with the aim of generalizing string automata. Recently, natural language researchers have been constructing weighted syntax-based models for problems
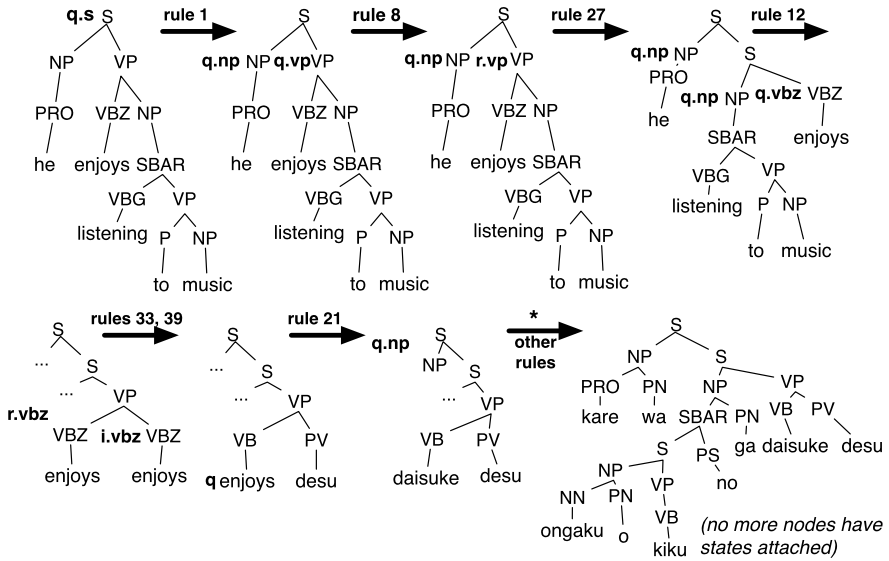
Fig. 6. Example of a syntax-based translation model [34]

such as language translation [61, 60, 1, 22, 16, 43], summarization [35], paraphrasing [48], question answering [14], and language modeling [5]. It has therefore become important to understand whether these natural language models can be captured by standard tree automata.

Figure 6 shows a syntax-based translation model that can be contrasted with the string-based model depicted in Figs. 2 and 3. In the upper left of the figure is an English tree, and in the lower right is a Japanese tree. In between, we see a top-down model of transformation in which pieces of English syntax are matched and replaced with pieces of Japanese syntax. Ultimately, individual English words and phrases are replaced with Japanese ones. This transformation can be carried out by a top-down tree transducer with ε-transitions, as defined by [51, 57], a fragment of which is shown in Fig. 8. This type of transducer is theoretically quite powerful, employing rules that copy unbounded pieces of input (as in Rule 4) and rules that delete pieces of input without processing them (as in Rule 34). It is well known that copying and deleting complicate matters—for example, the class of transformations induced by copying transducers is not closed under composition, which is a significant departure from the string case.

Figure 7 shows some natural transformations that arise in translating one human language to another. In the first example, an English noun-phrase (NP1) must be moved after the verb when we translate to Arabic. A standard noncopying tree transducer cannot handle this case, because it is necessary to "grab and reorder" structures that are deep in the input tree (such as VB and NP2), while the standard transducer can only get hold of the direct children
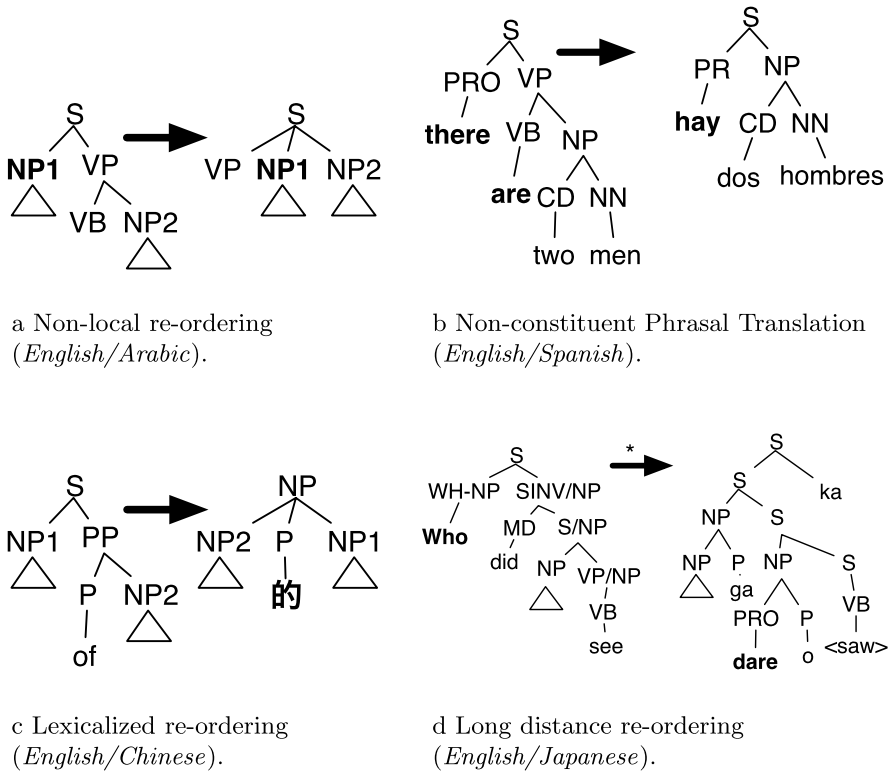
a Non-local re-ordering (*English/Arabic*).

b Non-constituent Phrasal Translation (*English/Spanish*).

c Lexicalized re-ordering (*English/Chinese*).

d Long distance re-ordering (*English/Japanese*).

Fig. 7. Examples of reordering made possible with a syntax-based translation model

of an input node. For this reason, [24] introduces a class of top-down tree transducers whose rules have extended left-hand sides. An example of such a rule is

```
q S(x0:NP VP(x1:VB x2:NP)) -> S(q x1, r x0, s x2)
```

In [19], Galley et al. give algorithms for acquiring such tree transducers from bilingual data. The English side of this data must be automatically parsed; this is typically done with statistical techniques such as in [10]. At the time of this writing, the largest such transducer has 500 million rules, and the empirical performance of the associated translation system compares favorably with string-based methods. Currently, work at the intersection of tree automata and natural language processing is active:

- On the empirical side, researchers aim to improve tree-based translation by building better models of translation and better rule-extraction algorithms. To further those goals, the availability of a toolkit for manipulating tree automata and tree transducers, such as [42], is important. Similar toolkits for string automata and transducers [46, 23] have enabled better

```
/* translate */

1. q.s S(x0, x1)          →^0.9 S(q.np x0, q.vp x1)
2. q.s S(x0, x1)          →^0.1 S(q.vp x1, q.np x0)
3. q.np x                 →^0.1 r.np x
4. q.np x                 →^0.8 NP(r.np x, i x)
5. q.np x                 →^0.1 NP(i x, r.np x)
6. q.pro PRO(x0)          →^1.0 PRO(q x0)
7. q.nn NN(x0)            →^1.0 NN(q x0)
8. q.vp x                 →^0.8 r.vp x
9. q.vp x                 →^0.1 S(r.vp x, i x)
10. q.vp x                →^0.1 S(i x, r.vp x)
11. q.vbz x               →^0.4 r.vbz x
12. q.vbz x               →^0.5 VP(r.vbz x, i x)
13. q.vbz x               →^0.1 VP(i x, r.vbz x)
14. q.sbar x              →^0.3 r.sbar x
15. q.sbar x              →^0.6 SBAR(r.sbar x, i x)
16. q.sbar x              →^0.1 SBAR(i x, r.sbar x)
17. q.vbg VBG(x0)         →^1.0 VP(VB(q x0))
18. q.pp PP(x0, x1)       →^1.0 NP(q.np x1, q.p x0)
19. q.p P(x0)             →^1.0 PN(q x0)
20. q he                  →^1.0 kare
21. q enjoys              →^0.1 daisuki
22. q listening           →^0.2 kiku
23. q to                  →^0.1 o
24. q to                  →^0.7 ni
25. q music               →^0.8 ongaku
26. r.vp VP(x0, x1)       →^0.9 S(q.vbz x0, q.np x1)
27. r.vp VP(x0, x1)       →^0.1 S(q.np x1, q.vbz x0)
28. r.sbar SBAR(x0, x1)   →^0.1 S(q.vbg x0, q.pp x1)
29. r.sbar SBAR(x0, x1)   →^0.9 S(q.pp x1, q.vbg x0)
30. r.np NP(x0)           →^0.1 q.pro x0
31. r.np NP(x0)           →^0.8 q.nn x0
32. r.np NP(x0)           →^0.1 q.sbar x0
33. r.vbz VBZ(x0)         →^0.7 VB(q x0)


/* insert */

34. i NP(x0)              →^0.3 PN(wa)
35. i NP(x0)              →^0.3 PN(ga)
36. i NP(x0)              →^0.2 PN(o)
37. i NP(x0)              →^0.1 PN(ni)
38. i SBAR(x0, x1)        →^0.7 PS(no)
39. i VBZ(x0)             →^0.2 PV(desu)
```

Fig. 8. Fragment of a top-down tree transducer with ε-transitions implementing a syntax-based translation model [34]

model development where the domain of strings is involved. Similarly, tree automata toolkits allow the reenvisioning of previous models in a clean transducer framework [25] as well as the rapid development of new models [41].

- On the algorithms side, researchers create more efficient procedures and data structures for executing tree-based inferences. For example, in [26], Huang and Chiang present efficient algorithms for extracting the k most probable trees from a context-free grammar, which is useful for extracting the k-best translations from a large set of hypotheses encoded as a grammar. In [24], Graehl and Knight give EM algorithms for training tree transducers. In [41], May and Knight show improvements from determinizing weighted tree automata. In [38], Maletti gives an $O(mn^4)$ algorithm for minimizing a weighted tree automaton with m rules and n states.
- On the theory side, researchers investigate which automata models both (1) fit natural language phenomena, and (2) possess good theoretical properties. There is still much work to be done—for example, transformations induced by extended left-hand side transducers are not closed under composition, even in the noncopying, nondeleting case. Researchers have also been investigating connections between tree automata and synchronous grammars [55]; the latter of which have been developed independently in the natural language community.

Another area of promise is syntax-based language modeling [5]. Here, we build a probability distribution over all English trees, rather than all English strings.[3] We hope to concentrate probability on objects that are grammatically well formed. Returning to our noisy-channel framework, we can then envision a language model represented by a regular tree grammar [20] and a channel model consisting of a cascade of tree transducers.

4.1 Open Problems

Knight and Graehl, in [34], presented a list of open problems pertinent to the use of weighted tree automata in natural language processing applications. Although some of those problems have since been at least partially solved (for example, in Tiburon [42] we have an instantiation of a useful, generic tree transducer toolkit, and the properties of extended tree transducers were studied in [39]) some still remain and new problems often arise, such as:

- Eppstein [17] presents an algorithm for finding the k best paths through a WFSA which runs in $O(m + n \log n + k)$ time, while [26] present an algorithm for finding the k best derivations of a weighted tree automaton which runs in $O(m + nk \log k)$ time. It is unknown whether the separation of k and n can be achieved in the tree case.

---

[3] We can still get the probability of a string by summing over all the trees who have that yield.

- Is there an algorithm for minimizing deterministic weighted tree automata that improves on the $O(mn^4)$ runtime of [38]?
- Is there an algorithm to determine whether two instances of a class of weighted tree transducers may be composed to form a single instance of that class that captures the same sequential transformation, even though the class itself is known to be not closed under composition?
- Is there a class of weighted tree transducers that (a) is expressive enough to capture natural language translation phenomena, (b) is closed under composition, (c) allows for unbounded output, (d) admits an algorithm for efficient weight training from input/output examples, and (e) preserves regularity?
- Can we cast other existing NLP models in the language of tree machinery, as was done in [25] for the model of [61], and can we extend these models in interesting ways?

5 Conclusion

In this chapter, we have surveyed some of the natural language applications in which weighted automata play a role. We expect the number of applications to grow over the coming years as automata theorists, linguists, and engineers collaborate to solve difficult problems, and as computational power grows to support new research avenues. Thatcher's vision, that we may use automata to solve problems, model behavior, and make advances in the natural language domain seems realized in current research efforts, and we have high hopes for the future.

References

1. H. Alshawi, S. Douglas, and S. Bangalore. Learning dependency translation models as collections of finite-state head transducers. *Computational Linguistics*, 26(1):45–60, 2000.
2. J.K. Baker. The DRAGON system—An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-23(1):24–29, 1975.
3. T. Brants, A.C. Popat, P. Xu, F.J. Och, and J. Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Prague, Czech Republic, June 2007*, pages 858–867. Association for Computational Linguistics, Stroudsburg, 2007.
4. P.F. Brown, S.A.D. Pietra, V.J.D. Pietra, and R.L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–312, 1993.

5. E. Charniak. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, Toulouse, France, July 2001*, pages 116–123. Association for Computational Linguistics, Stroudsburg, 2001.

6. N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

7. N. Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.

8. K.W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Second Conference on Applied Natural Language Processing Proceedings, Austin, TX, February 1988*, pages 136–143. Association for Computational Linguistics, Stroudsburg, 1988.

9. A. Clark. Memory-based learning of morphology with stochastic transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, July 2002*, pages 513–520. Association for Computational Linguistics, Stroudsburg, 2002.

10. M. Collins. Head-driven statistical models for natural language parsing. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1999.

11. M. Dalrymple. *Lexical Functional Grammar*. Academic Press, New York, 2001.

12. A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

13. R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

14. A. Echihabi and D. Marcu. A noisy-channel approach to question answering. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics, Sapporo, Japan, July 2003*, pages 16–23. Association for Computational Linguistics, Stroudsburg, 2003.

15. S. Eilenberg. *Automata, Languages, and Machines*. Academic Press, New York, 1974.

16. J. Eisner. Learning non-isomorphic tree mappings for machine translation. In *The Companion Volume to the Proceedings of 41st Annual Meeting of the Association for Computational Linguistics, Sapporo, Japan, July 2003*, pages 205–208. Association for Computational Linguistics, Stroudsburg, 2003.

17. D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

18. R.A. Fisher. On the "probable error" of a coefficient of correlation deduced from a small sample. *Metron. International Journal of Statistics*, 1:3–32, 1921.

19. M. Galley, M. Hopkins, K. Knight, and D. Marcu. What's in a translation rule? In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT–NAACL 2004, Boston, MA, May 2004*, pages 273–280. Association for Computational Linguistics, Stroudsburg, 2004.

20. F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

21. F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, *volume 3*, Chapter 1, pages 1–68. Springer, Berlin, 1997.

22. D. Gildea. Loosely tree-based alignment for machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, *Sapporo, Japan*, *July 2003*, pages 80–87. Association for Computational Linguistics, Stroudsburg, 2003.

23. J. Graehl. Carmel finite-state toolkit. http://www.isi.edu/licensed-sw/carmel, 1997.

24. J. Graehl and K. Knight. Training tree transducers. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT–NAACL 2004*, *Boston, MA*, *May 2004*, pages 105–112. Association for Computational Linguistics, Stroudsburg, 2004.

25. J. Graehl, K. Knight, and J. May. Training tree transducers. *Computational Linguistics*, 34(3):391–427, 2008.

26. L. Huang and D. Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, *Vancouver, Canada*, *October 2005*, pages 53–64. Association for Computational Linguistics, Stroudsburg, 2005.

27. E.T. Jaynes. Information theory and statistical mechanics. *Physical Review (Series II)*, 106(4):620–630, 1957.

28. F. Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, 1976.

29. F. Jelinek, L.R. Bahl, and R.L. Mercer. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, IT-21(3):250–256, 1975.

30. D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd edition. Chapter 4: N-grams. Prentice Hall, Englewood Cliffs, 2009.

31. R. Kaplan and M. Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

32. K. Knight and Y. Al-Onaizan. Translation with finite-state devices. In *Machine Translation and the Information Soup: Third Conference of the Association for Machine Translation in the Americas, AMTA'98*, *Langhorne, PA*, *October 1998*, volume 1529 of *Lecture Notes in Computer Science*, pages 421–437. Springer, Berlin, 1998.

33. K. Knight and J. Graehl. Machine transliteration. *Computational Linguistics*, 24(4):599–612, 1998.

34. K. Knight and J. Graehl. An overview of probabilistic tree transducers for natural language processing. In *Computational Linguistics and Intelligent Text Processing 6th International Conference, CICLing 2005*, *Mexico*

City, Mexico, February 2005, volume 3406 of *Lecture Notes in Computer Science*, pages 1–24. Springer, Berlin, 2005.

35. K. Knight and D. Marcu. Summarization beyond sentence extraction: A probabilistic approach. *Artificial Intelligence*, 139(1):91–107, 2002.

36. O. Kolak, W. Byrne, and P. Resnik. A generative probabilistic OCR model for NLP applications. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, Edmonton, Canada, May–June 2003*, pages 55–62. Association for Computational Linguistics, Stroudsburg, 2003.

37. S. Kumar and W. Byrne. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, Edmonton, Canada, May–June 2003*, pages 63–70. Association for Computational Linguistics, Stroudsburg, 2003.

38. A. Maletti. Minimizing deterministic weighted tree automata. In *Proceedings of the 2nd International Conference on Language and Automata Theory and Applications*, pages 371–382. Universitat Rovira I Virgili, Tarragona, 2008.

39. A. Maletti, J. Graehl, M. Hopkins, and K. Knight. The power of extended top-down tree transducers. *SIAM Journal on Computing*, 39(2):410–430, 2009.

40. A.A. Markov. Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain (Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, 7:153–162, 1913. English translation by Morris Halle, 1956.

41. J. May and K. Knight. A better n-best list: Practical determinization of weighted finite tree automata. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference, New York, NY, June 2006*, pages 351–358. Association for Computational Linguistics, Stroudsburg, 2006.

42. J. May and K. Knight. Tiburon: A weighted tree automata toolkit. In O.H. Ibarra and H.-C. Yen, editors, *Proceedings of the 11th International Conference of Implementation and Application of Automata, CIAA 2006, Taipei, Taiwan, August 2006*. volume 4094 of *Lecture Notes in Computer Science*, pages 102–113. Springer, Berlin, 2006.

43. I.D. Melamed. Multitext grammars and synchronous parsers. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, Edmonton, Canada, May–June 2003*, pages 79–86. Association for Computational Linguistics, Stroudsburg, 2003.

44. G.A. Miller and N. Chomsky. Finitary models of language users. In R.D. Luce, R.R. Bush, and E. Galanter, editors, *Handbook of Mathematical Psychology, volume II*, pages 419–491. Wiley, New York, 1963.

45. M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–312, 1997.

46. M. Mohri, F.C.N. Pereira, and M.D. Riley. AT&T FSM library. http://www.research.att.com/~fsmtools/fsm, 1998. AT&T Labs—Research.

47. F. Och, C. Tillmann, and H. Ney. Improved alignment models for statistical machine translation. In *Proceedings of the 1999 Joint SIGDAT Conference of Empirical Methods in Natural Language Processing and Very Large Corpora, College Park, MD, June 1999*, pages 20–28. Association for Computational Linguistics, Stroudsburg, 1999.

48. B. Pang, K. Knight, and D. Marcu. Syntax-based alignment of multiple translations: Extracting paraphrases and generating new sentences. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, Edmonton, Canada, May–June 2003*, pages 102–109. Association for Computational Linguistics, Stroudsburg, 2003.

49. F. Pereira, M. Riley, and R. Sproat. Weighted rational transductions and their application to human language processing. In *Human Language Technology, Plainsboro, NJ, March 1994*, pages 262–267. Morgan Kaufmann, San Mateo, 1994.

50. M. Riley, F. Pereira, and E. Chun. Lazy transducer composition: A flexible method for on-the-fly expansion of context-dependent grammar network. In *Proceedings, IEEE Automatic Speech Recognition Workshop, Snowbird, UT, December 1995*, pages 139–140.

51. W.C. Rounds. Mappings and grammars on trees. *Theory of Computing Systems*, 4:257–287, 1970.

52. I.A. Sag, T. Wasow, and E.M. Bender. *Syntactic Theory*, 2nd edition. CSLI Publications, Stanford, 2003.

53. C. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948. 623–656

54. S.M. Shieber. Synchronous grammars as tree transducers. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+ 7), Vancouver, Canada, May 2004*, pages 88–95.

55. S.M. Shieber. Unifying synchronous tree adjoining grammars and tree transducers via bimorphisms. In *11th Conference of the European Chapter of the Association for Computational Linguistics, Trento, Italy, April 2006*, pages 377–384. Association for Computational Linguistics, Stroudsburg, 2006.

56. R. Sproat, W. Gales, C. Shih, and N. Chang. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3):377–404, 1996.

57. J.W. Thatcher. Generalized^2 sequential machine maps. *Journal of Computer and System Sciences*, 4(4):339–367, 1970.

58. J.W. Thatcher. Tree automata: An informal survey. In A.V. Aho, editor, *Currents in the Theory of Computing*, pages 143–172. Prentice Hall, Englewood Cliffs, 1973.

59. W.A. Woods. Transition network grammars for natural language analysis. *Communications of the Association for Computing Machinery*, 13(10):591–606, 1970.

60. D. Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404, 1997.

61. K. Yamada and K. Knight. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, Toulouse, France, July 2001*, pages 523–530. Association for Computational Linguistics, Stroudsburg, 2001.

62. D. Zajic, B. Dorr, and R. Schwartz. Automatic headline generation for newspaper stories. In *Proceedings of the ACL-02 Workshop on Text Summarization (DUC 2002), Philadelphia, PA, July 2002*, pages 78–85. Association for Computational Linguistics, Stroudsburg, 2002.

63. B. Zhou, S.F. Chen, and Y. Gao. Folsom: A fast and memory-efficient phrase-based approach to statistical machine translation. In *Proceedings of the IEEE/ACL 2006 Workshop on Spoken Language Technology, Palm Beach, Aruba, December 2006*, pages 226–229, IEEE Press, New York, 2006.