Applications of Weighted Automata in Natural Language Processing

Kevin Knight[1] and Jonathan May[2]

[1] USC Information Sciences Institute, 4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292, USA knight@isi.edu
[2] USC Information Sciences Institute, 4676 Admiralty Way, Suite 1001, Marina del Rey, CA 90292, USA 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 [35, 27] to predict sequences of vowels and consonants in novels by Pushkin. Shannon [48] 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* (TG), loosely defined in [7]. In attempting to formalize TG, automata theorists like Rounds [46] and Thatcher [52] 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. In the final paragraph of his 1973 tree automata survey [53], 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, 18], while linguists went the other way and eschewed formalism. Computational linguistics focused for a time on extensions to CFG [11, 47], 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 [26, 25, 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 [20, 41].

The 21st century has seen a re-awakened interest in tree automata among computational linguists [31, 49, 22], particularly for problems like automatic language translation, where transformations are sensitive to syntactic structure. Generic tree automata toolkits [37] 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 [30]. As a first attempt, we might contemplate a single finite-state transducer that converts streams of Katakana symbols K into streams of English letters E with a corresponding probability of conversion $P(E|K)$ and chooses the most likely E:

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

The corresponding transducer design looks like this:

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

$$\underset{E}{\operatorname{argmax}} P(E|K) = \underset{E}{\operatorname{argmax}} \frac{P(E)P(K|E)}{P(K)} \tag{2}$$

$$= \underset{E}{\operatorname{argmax}} P(E)P(K|E) \tag{3}$$

The corresponding transducer design now looks like this:

$\boxed{\text{WFSA}}$ $\Rightarrow$ English $\Rightarrow$ $\boxed{\text{WFST}}$ $\Rightarrow$ 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 backwards through the WFST, which will produce a multiplicity of English phrases that would transduce to K. (These English phrases can be represented as a finite-state acceptor, since the WFST preserves regularity in both directions). 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* [48], 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 non-deterministically 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.

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 [30]:

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

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.

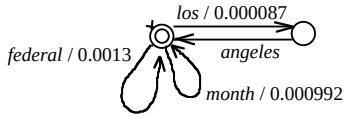
We justify this design in our probability model by using the conditional probability chain rule to break one probability distribution into a chain of independent distributions:

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

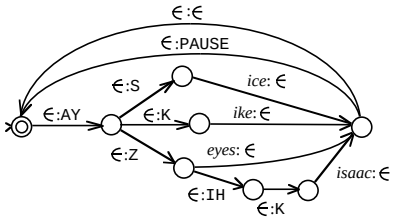
where C is any newly introduced parameter space that we can sum over.

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

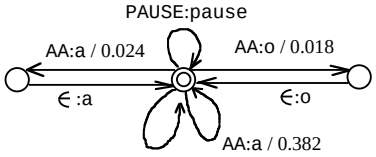
The probability model equation then becomes:



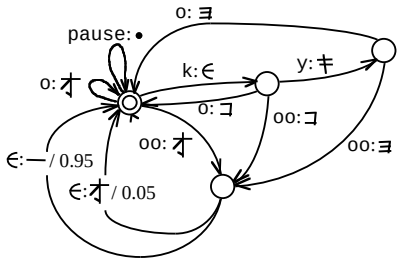
a WFSA A: Produce an English phrase



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 [30]

$$\begin{aligned} &\underset{E}{\operatorname{argmax}}\, P(E|K) = \\ &\underset{E}{\operatorname{argmax}} \sum_{es} \sum_{js} \quad P(E) \cdot P(es|E) \cdot \\ &\qquad\qquad\qquad\qquad P(js|es) \cdot P(K|js) \end{aligned} \tag{5}$$

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—Figure 1 shows fragments. WFSA A (Figure 1a) non-deterministically generates English word sequences. WFST B (Figure 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 (Figure 1c) converts English sounds into Japanese sounds. This is a highly

non-deterministic 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 non-deterministic in the reverse direction as well, since a Japanese *r* sound may transduce to an English *R* or *L*. However, the WFST can make its substitutions largely independent of context. Finally, WFST D (Figure 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 backwards through WFST D, then sending the result (which itself can be represented by a WFSA) backwards 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
```

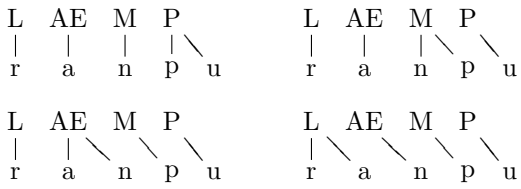
Here is where the probabilities from WFSA A and WFST C are important. If these are set to reflect what happens in the world (i.e., which English phrases are popular, and which sound substitutions are popular), then each potential English output also comes with a score, and we can ask for the top-scoring 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 compose automata A, B, C, and D into a single machine, offline, then determinize and/or minimize it for deployment [40]. A third approach is to employ lazy composition [45, 57], which executes a parallel search through K, A, B, C, and D 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 Equation 5. Each of the factors in Equation 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 Figure 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):



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:

$$P(js|es) = \sum_{a} \prod_{i=1}^{|es|} P(j_{seq_{es_i}}|es_i) \tag{6}$$

where alignment a maps each English sound es_i onto Japanese sound subsequence $j_{seq_{es_i}}$.

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:

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, 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:

and many other pairs.

Given any particular set of parameter values, such as $P(n|M) = 0.32$ (and so on), we can compute $P(j|e)$ 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 [24] 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 re-ordering. 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:

Spanish ⇒ WFST ⇒ 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 Section 2.1:

WFSA ⇒ English Sentence ⇒ WFST ⇒ Spanish Sentence

In this scheme, the WFST can operate in a largely context-independent fashion. Sending a particular Spanish sentence backwards 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.
```

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

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? [4] proposed a particular model for $P(s|e)$ which would assign a conditional probability to any pair of Spanish and English strings. [29] cast this model as a sequence of finite-state automata. Figure 2 depicts the operation of these automata, and Figure 3 shows automata fragments.

WFSA A (Figure 3a) generates English word sequences according to some probability distribution that (we hope) assigns high probability to grammatical, sensible sequences. WFST B (Figure 3b) decides, for each English word, whether to drop it, copy it, duplicate it, triplicate it, etc. The decision is based only on the word itself, with no context information. After each resulting word, WFST C (Figure 3c) inserts a NULL token with probability 0.02. WFST D (Figure 3d) then translates each word, one for one, into Spanish. (The NULL word is designed to generate Spanish function words that have no English equivalent). Finally, WFST E (Figure 3e) re-orders the resulting Spanish words. Each transducer is simple enough to build; all of them are highly non-deterministic.

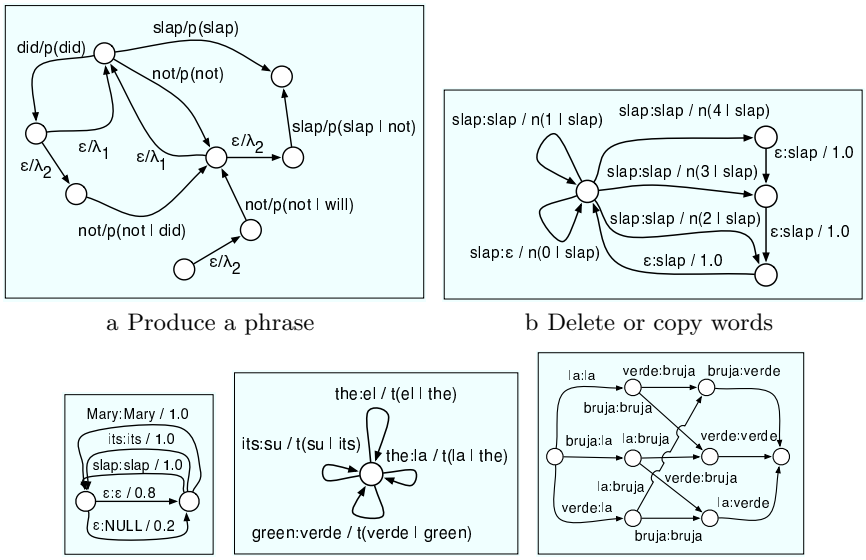
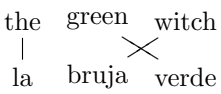


Fig. 3. Five automata fragments that translate Spanish to English. WFSA A produces an English phrase, and WFSTs B-E transform that phrase into Spanish.

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 backwards through WFST E, to obtain various re-orderings, 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.:



then we could estimate the parameters of WFSTs B, C, D, and E. For example, out of 1000 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 Figure 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 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 re-orderings 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 cast in terms of tree automata rather than string automata, 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 [48] observed that the generation of natural language text could be approximated to a reasonable measure by a WFSA that uses state 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 Figure 4.

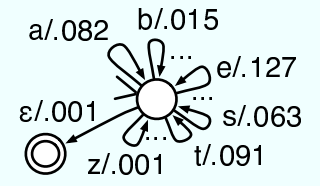


Fig. 4. WFSA for a 1-gram letter language model

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 $\mathrm{P}(e \mid r)$. 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, 39]. 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* re-assigns some of the probability from seen events to unseen events. One simple technique is *interpolation* smoothing. For the 2-gram case, instead of estimating $P(y|x)$ as $\frac{count(xy)}{count(x)}$, which might be zero, we interpolate with the 1-gram probability of y:

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

where N is the size of the training corpus. Likewise, $P(z|x,y)$ can be estimated as $\lambda_2 \frac{count(xyz)}{count(xy)} + (1-\lambda_2)P(z|y)$. 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 Figure 5. This formulation is space efficient, requiring only one transition per observed n-gram, rather than one transition per conceivable n-gram.

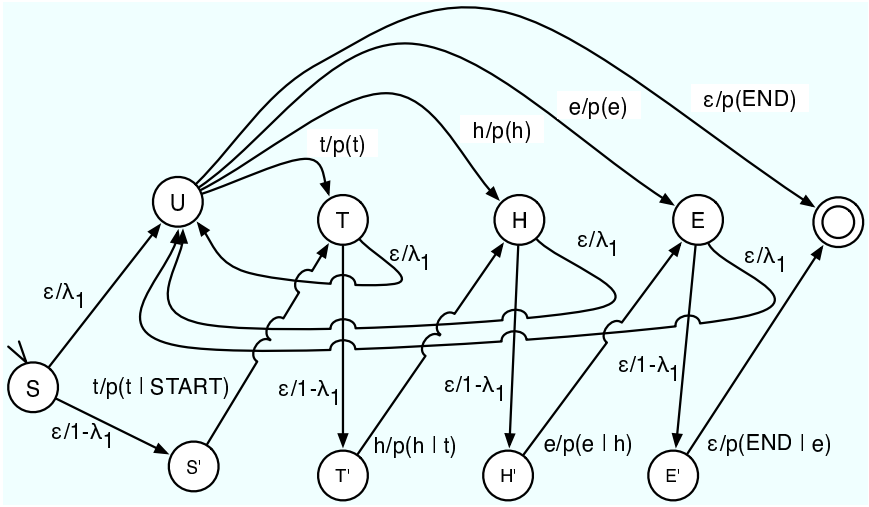


Fig. 5. Fragment of a WFSA for a bigram letter language model. At each state {S, T, H, E} a decision is made to use bigram context by moving to states {S', T', H', E'}, respectively, or to use unigram context by moving to state U.

3 Applications of Weighted String Automata

In Section 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.

3.1 Language Translation

We described a word-for-word model of language translation in Section 2. This model was implemented in a WFST framework by [29]. A phrase-for-phrase model was subsequently devised by [42] and implemented in a WFST framework by [34]. Translations from this model are much more accurate, and by using a WFST toolkit, [34] 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 re-ordering—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 re-orderings online as a temporary WFSA, which is then sent through the rest of the noisy-channel cascade.

3.2 Speech Recognition

[44] 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 Section 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 [41].

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 white-space, such as Chinese. [51] 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 (Section 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 [28], 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 [56], 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. [56] 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 non-digital manner, e.g. converting hand-written 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. [33] use this approach, and built their final system with the AT&T FSM toolkit [41], thus using automata operations directly. The chain of transducers in this case first segments the words into characters, then groups the characters into sub-word sequences, and finally transforms the sequences into noise-filled sequences.

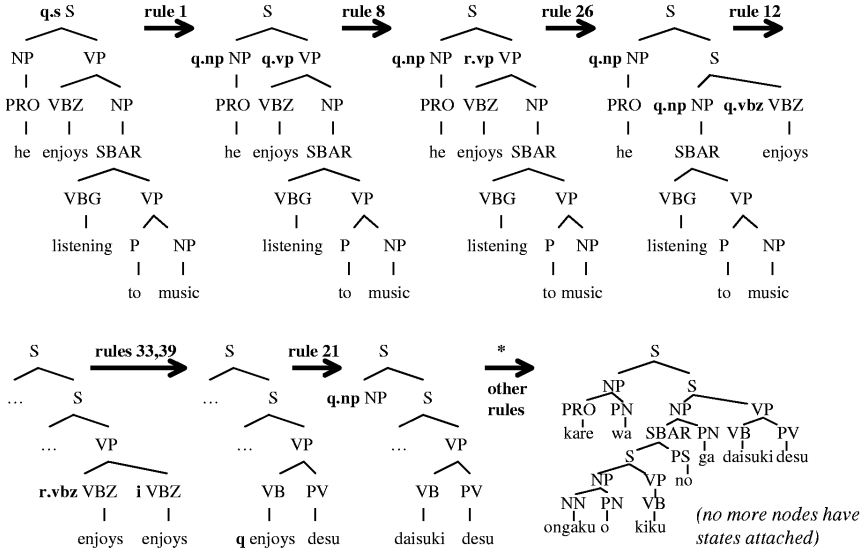


Fig. 6. Example of a syntax-based translation model [31]

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 re-ordering 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 [46, 52] with the aim of generalizing string automata. Recently, natural language researchers have been constructing weighted syntax-based models for problems such as language translation [55, 54, 1, 19, 16, 38], summarization [32], paraphrasing [43], 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 Figures 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, as defined by [46, 52], a fragment of which is shown in Figure 8. This type of transducer

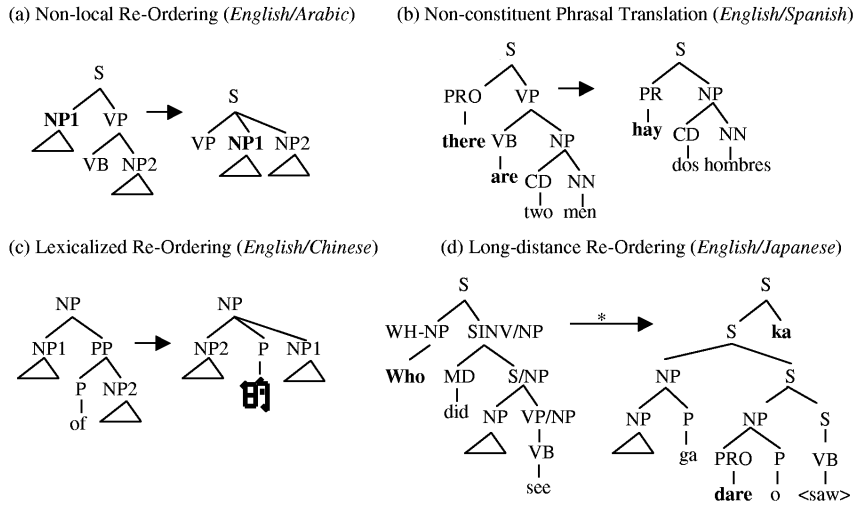


Fig. 7. Examples of reordering made possible with a syntax-based translation model

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 non-copying tree transducer cannot handle this case, because it is necessary to "grab and re-order" 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 of an input node. For this reason, [21] 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)
```

[17] gives 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 [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:

```
/* 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 implementing a syntax-based translation model [31]

- 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 [37], is important. Similar toolkits for string automata and transducers [41, 20] have enabled better model development where the domain of strings is involved. Similarly, tree automata toolkits allow the re-envisioning of previous models in a clean transducer framework [22] as well as the rapid development of new models [36].
- On the algorithms side, researchers create more efficient procedures and data structures for executing tree-based inferences. For example, [23] 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. [21] give EM algorithms for training tree transducers. [36] show improvements from determinizing tree automata.
- 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 non-copying, non-deleting case. Researchers have also been investigating connections between tree automata and synchronous grammars [50], 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. We hope to concentrate probability on objects that are grammatically well-formed. (We can still get the probability of a string by summing over all the trees who have that yield). Returning to our noisy-channel framework, we can then envision a language model represented by a regular tree grammar [18] and a channel model consisting of a cascade of tree transducers.

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. Hiyan Alshawi, Shona Douglas, and Srinivas Bangalore. Learning dependency translation models as collections of finite-state head transducers. *Computational Linguistics*, 26(1), March 2000.
2. James K. Baker. The DRAGON system – An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-23(1):24–29, February 1975.
3. Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey 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*, pages 858–867, Prague, June 2007.
4. Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–312, June 1993.
5. Eugene Charniak. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 116–123, 2001.
6. Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.
7. Noam Chomsky. *Syntactic Structures*. Mouton, 1957.
8. Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Second Conference on Applied Natural Language Processing Proceedings*, pages 136–143, Austin, Texas, 1988.
9. Alexander Clark. Memory-based learning of morphology with stochastic transducers. In *ACL Proceedings*, 2002.
10. Michael Collins. *Head-driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, Philadelphia, 1999.
11. Mary 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), 1977.
13. Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley, New York, 1973.
14. Abdessamad Echihabi and Daniel Marcu. A noisy-channel approach to question answering. In *ACL Proceedings*, 2003.
15. Samuel Eilenberg. *Automata, languages, and machines*. Academic Press, New York, 1974-76.
16. Jason Eisner. Learning non-isomorphic tree mappings for machine translation. In *ACL Proceedings (companion volume)*, Sapporo, Japan, 2003.
17. Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In *HLT-NAACL Proceedings*, 2004.
18. Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.
19. Daniel Gildea. Loosely tree-based alignment for machine translation. In *ACL Proceedings*, Sapporo, Japan, 2003.
20. Jonathan Graehl. Carmel finite-state toolkit. http://www.isi.edu/licensed-sw/carmel, 1997.
21. Jonathan Graehl and Kevin Knight. Training tree transducers. In *HLT-NAACL Proceedings*, 2004.

22. Jonathan Graehl, Kevin Knight, and Jonathan May. Training tree transducers. *Computational Linguistics*, To Appear.
23. Liang Huang and David Chiang. Better k-best parsing. In *IWPT Proceedings*, 2005.
24. E. T. Jaynes. Information theory and statistical mechanics. *Physical Review (Series II)*, 106(4):620–630, May 1957.
25. Frederick Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, April 1976.
26. Frederick Jelinek, Lalit R. Bahl, and Robert L. Mercer. Design of a linguistic statistical decoder for the recognition of continuous speech. *IEEE Transactions on Information Theory*, IT-21(3):250–256, May 1975.
27. Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, chapter Chapter 6: N-grams. Prentice Hall, 2000.
28. Ronald Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 1994.
29. Kevin Knight and Yasser Al-Onaizan. Translation with finite-state devices. In *AMTA Proceedings*, 1998.
30. Kevin Knight and Jonathan Graehl. Machine transliteration. *Computational Linguistics*, 24(4):599–612, December 1998.
31. Kevin Knight and Jonathan Graehl. An overview of probabilistic tree transducers for natural language processing. In *CICLing Proceedings*, 2005.
32. Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: A probabilistic approach. *Artificial Intelligence*, 139, 2002.
33. Okan Kolak, William Byrne, and Philip Resnik. A generative probabilistic OCR model for NLP applications. In *Proceedings of HLT-NAACL*, pages 55–62, Edmonton, May-June 2003.
34. Shankar Kumar and William Byrne. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *HLT-NAACL Proceedings*, 2003.
35. 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.
36. Jonathan May and Kevin 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*, pages 351–358, New York City, USA, June 2006. Association for Computational Linguistics.
37. Jonathan May and Kevin Knight. Tiburon: A weighted tree automata toolkit. In Oscar H. Ibarra and Hsu-Chun Yen, editors, *Proceedings of the 11th International Conference of Implementation and Application of Automata, CIAA 2006*, volume 4094 of *Lecture Notes in Computer Science*, pages 102–113, Taipei, Taiwan, August 2006. Springer.
38. I. Dan Melamed. Multitext grammars and synchronous parsers. In *NAACL Proceedings*, 2003.
39. 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. John Wiley, New York, 1963.

40. Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–312, June 1997.
41. Mehryar Mohri, Fernando C. N. Pereira, and Michael D. Riley. AT&T FSM library. http://www.research.att.com/fsmtools/fsm, 1998. AT&T Labs – Research.
42. F. Och, C. Tillmann, and H. Ney. Improved alignment models for statistical machine translation. In *EMNLP Proceedings*, 1999.
43. Bo Pang, Kevin Knight, and Daniel Marcu. Syntax-based alignment of multiple translations: Extracting paraphrases and generating new sentences. In *NAACL-HLT Proceedings*, 2003.
44. Fernando Pereira, Michael Riley, and Richard Sproat. Weighted rational transductions and their application to human language processing. In *HLT Proceedings*, 1994.
45. Michael Riley, Fernando Pereira, and Emerald Chung. Lazy transducer composition: A flexible method for on-the-fly expansion of context-dependent grammar network. In *Proceedings, IEEE Automatic Speech Recognition Workshop*, Snowbird, Utah, December 1995.
46. William C. Rounds. Mappings and grammars on trees. *Mathematical Systems Theory*, 4:257–287, 1970.
47. Ivan A. Sag, Thomas Wasow, and Emily M. Bender. *Syntactic Theory.* CSLI Publications, 2nd edition, 2003.
48. Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27, 1948.
49. Stuart M. Shieber. Synchronous grammars as tree transducers. In *TAG+ Proceedings*, 2004.
50. Stuart M. Shieber. Unifying synchronous tree adjoining grammars and tree transducers via bimorphisms. In *EACL Proceedings*, 2006.
51. Richard Sproat, William Gales, Chilin Shih, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3), September 1996.
52. James W. Thatcher. Generalized^2 sequential machines. *Journal of Computer System Science*, pages 339–367, 1970.
53. James 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, NJ, 1973.
54. Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23:377–404, 1997.
55. Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *ACL Proceedings*, pages 523–530, 2001.
56. David Zajic, Bonnie Dorr, and Richard Schwartz. Automatic headline generation for newspaper stories. In *Workshop on Automatic Summarization Proceedings*, 2002.
57. Bowen Zhou, Stanley F. Chen, and Yuqing 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*, pages 226–229, Palm Beach, Aruba, December 10–13 2006.

Index