A Weighted Tree Transducer Toolkit for Syntactic Natural Language Processing Models

Jonathan May

June 2008

Abstract

Solutions for many natural language processing problems such as speech recognition, transliteration, and translation have been described as weighted finite-state transducer cascades. The transducer formalism is very useful for researchers, not only for its ability to expose the deep similarities between seemingly disparate models, but also because expressing models in this formalism allows for rapid implementation of real, data-driven systems. Finite-state toolkits can interpret and process transducer chains using generic algorithms and many real-world systems have been built using these toolkits. Current research in NLP makes use of syntax-rich models that are poorly suited to extant transducer toolkits, which process linear input and output. Tree transducers can handle these models, and a weighted tree transducer toolkit with appropriate generic algorithms will lead to the sort of gains in syntax-based modeling that were achieved with string transducer toolkits. This thesis presents Tiburon, a weighted tree transducer toolkit that is highly suited for the processing of syntactic NLP models. Tiburon contains implementations of novel determinization, minimization, and training algorithms that are extensions of classical algorithms suitable for the processing of weighted tree transducers and automata. Tiburon also contains algorithms reflecting previously known results for generation, composition, and projection. We show Tiburon's effectiveness at rapid reproduction of previously presented syntax-based machine translation and parsing models as well as its utility in constructing and evaluating novel models.

Chapter 1

Introduction: Models and Transducers

I can't work without a model.

Vincent Van Gogh

1.1 A cautionary tale

One of the earliest examples of a syntactic parser was built in 1958 and 1959 to run on the Univac 1 computer at the University of Pennsylvania and attempts were made to recreate the parser some forty years hence on modern machines (Joshi and Hopely, 1996). Given that computer science was in its infancy at the time of the parser's creation and much had changed in the interim, it is not surprising that the resurrectors relied on the hundreds of pages of flowcharts and program specifications describing the system as guidance, rather than the original assembly code itself. Unfortunately, due to ambiguities in the documentation, some guesses had to be made in reimplementation, and thus the reincarnation of the parser is only an approximation of the original system at best. As the resurrectors were faithful to the design of the original parser, however, they built the modern incarnation of the parser as a single piece of code designed to do a single task. Of course, this time they wrote the program in C rather than assembly. If, forty years from now, a new generation of computer science archaeologists wishes to re-recreate the parser, one hopes the C language is still understandable, and that the source code survives. Without this knowledge the team will have to make do with the documentation of the original system as well as the fourteen or so pages of description of the current reimplementation that document the assumptions made in the new code and hope that this serves to remove all ambiguity and that no undocumented features were added to the modern source. I weep for the future.

1.2 Transducers to the rescue

As it happens, this parser, now called Uniparse, was designed as a cascade of *finite-state transducers*, abstract machines that read an input tape and write an output tape based on a set of state transition rules. Transducers have been widely studied and have numerous attractive properties; among them is the property of *closure under composition*—the transformations accomplished by a sequence of two transducers can be captured by a single transducer. These properties, along with effective algorithms that take advantage of them, such as an algorithm to quickly construct the composition of two transducers, allow any program written in the form of a transducer cascade, as Uniparse was designed, to be easily and effectively handled by a generic program that processes and manipulates transducers.

Rather than writing Uniparse from the original design schematics in custom assembly or C, the resurrectors could have encoded the schematics themselves, which are already written as transducers, into a uniform format that is suitable for reading by a generic finite-state transducer toolkit and used transducer operations such as composition and *projection* (the obtaining of either the input or output language of a transducer)

to perform the parsing operations, rather than writing new code. By expressing Uniparse as transducer rules the resurrectors would have been able to combine the transducer design with its code implementation. Any alterations to the original design would have been encoded in the set of transducer rules, preventing any "hacks" from hiding in the code base. A file containing solely transducer rules requires no additional documentation to explain any hidden implementation, as the entire implementation is represented in the rules. Most importantly, aside from formatting issues, a file of transducer rules is immune from the havoc time wreaks on compiled code. Future generations could use their transducer toolkits on these files with a trivial number of changes.

Of course, someone has to build the transducer toolkit itself. And this naturally raises the question: Is implementing a program for finite-state transducers rather than for a syntactic parser simply trading one specific code base for another? Thankfully, transducer cascades are useful for more than light deterministic parsing. In the fields of phonological and morphological analysis Kaplan and Kay (1994) realized the analysis rules linguists developed could be encoded as finite-state transducer rules, and this led first to a transducer-based morphological analysis system (Koskenniemi, 1983) and eventually to the development of XFST (Karttunen, Gaál, and Kempe, 1997), an entire finite-state toolkit complete with the requisite algorithms needed to manipulate transducers and actually get work done. The set of natural language transformation tasks capturable by regular expressions such as date recognition, word segmentation, some simple part-of-speech tagging, and spelling correction, and light parsing such as that done by Uniparse can be expressed as cascades of finite-state transducers (Karttunen et al., 1997). The availability of this toolkit allowed researchers to simply write their models down as transducer rules and allow the toolkit to do the processing work.

1.3 A transducer model of translation

Imagine we want to build a small program that translates between Spanish and English. Here is a simple model of how an English sentence becomes a Spanish sentence:

- An English sentence is imagined by someone fluent in English.
- The words of the sentence are rearranged—each word can either remain in its original position or swap places with the subsequent word.
- Each word is translated into a single Spanish word.

Such a model obviously does not capture all translations between Spanish and English, but it does handle some of them, and thus provides a good motivating example. Having envisioned this model of machine translation, an enterprising student could set about writing a program from scratch that implements the model. However, rather than enduring lengthy coding and debugging sessions the student could instead represent the model by the following cascade of finite-state transducers which may be composed into a single transducer using a toolkit such as XFST:

- A finite-state *automaton A*, which recognizes the sentences of English. This is a special case of a finite-state transducer—one where each rule has a single symbol rather than separate reading and writing symbols. A simple automaton for English maintains a state associated with the last word it saw, and only has transitions for valid subsequent words. Let us semantically associate the state q_x with cases where the last recognized word was x. Let q_{START} be the state representing the beginning of a sentence. We write rules like:
  - $q_{START} \xrightarrow{\text{the}} q_{the}$
  - $q_{the} \xrightarrow{\text{green}} q_{green}$
  - $q_{the} \xrightarrow{\text{ball}} q_{ball}$
  - $q_{green} \xrightarrow{\text{ball}} q_{ball}$
  - $q_{green} \xrightarrow{\text{horse}} q_{horse}$
  - ...

  and so on. This automaton will allow phrases such as "the ball" and "the green horse" but not "green ball the" or "horse green".

- A reordering transducer B with rules of the following form for all words a and b:
  - $r \xrightarrow{\text{a:}\epsilon} r_a$
  - $r \xrightarrow{\text{a:a}} r$
  - $r_a \xrightarrow{\text{b:b a}} r$

  Here, ϵ on the right side means no symbol is written when the rule is invoked, though the symbol on the left is consumed. The rules are instantiated for each possible pair of words, so given the English vocabulary {the, ball, green} we would have:

  - $r \xrightarrow{\text{the:}\epsilon} r_{the}$
  - $r \xrightarrow{\text{the:the}} r$
  - $r_{the} \xrightarrow{\text{ball:ball the}} r$
  - $r_{the} \xrightarrow{\text{green:green the}} r$
  - $r_{the} \xrightarrow{\text{the:the the}} r$
  - $r \xrightarrow{\text{ball:}\epsilon} r_{ball}$
  - $r \xrightarrow{\text{ball:ball}} r$
  - $r_{ball} \xrightarrow{\text{ball:ball ball}} r$
  - $r_{ball} \xrightarrow{\text{green:green ball}} r$
  - $r_{ball} \xrightarrow{\text{the:the ball}} r$
  - $r \xrightarrow{\text{green:}\epsilon} r_{green}$
  - $r \xrightarrow{\text{green:green}} r$
  - $r_{green} \xrightarrow{\text{ball:ball green}} r$
  - $r_{green} \xrightarrow{\text{green:green green}} r$
  - $r_{green} \xrightarrow{\text{the:the green}} r$

- A one-state transducer C that translates between English and Spanish, e.g.:
  - $s \xrightarrow{\text{ball:pelota}} s$
  - $s \xrightarrow{\text{the:el}} s$
  - $s \xrightarrow{\text{the:la}} s$
  - $s \xrightarrow{\text{horse:caballo}} s$
  - $s \xrightarrow{\text{green:verde}} s$
  - ...

These three transducers can be composed together using classic algorithms to form a single translation machine. A candidate Spanish phrase can then be encoded as a simple automaton D like so:

- q_0
- $q_0 \xrightarrow{\text{la}} q_1$
- $q_1 \xrightarrow{\text{pelota}} q_2$
- $q_2 \xrightarrow{\text{verde}} q_3$

This automaton represents exactly the phrase "la pelota verde". It can be composed to the right of the translation chain. Then the domain projection of the composed transducer can be taken to obtain the resulting translation, which in this case is "the green ball." Notice that B allows many phrases to ultimately translate to the Spanish phrase, but A restricts the domain to its language, which is valid English. By separately encoding transducers that translate between English and Spanish appropriately and transducers that recognize valid English we are able to impose both constraints.

There are many problems with this translation model. One of the most obvious is that we cannot handle cases where the number of English and Spanish words is not the same, so the translation between "I do not have" and "No tengo" is impossible. However, successive refinements and introductions of additional transducers, some with ϵ-rules, can help make the system better. Implementing the same refinements and model changes in a custom code implementation can require many more tedious coding and debugging cycles.

There is a more fundamental problem with this model that cannot easily be resolved by reconfiguring the transducers. If there are multiple valid answers, how do we know which to choose? In this framework a transformation is either correct or incorrect; there is no room for preference. We are faced with the choice of either overproducing and not knowing which of several answers is correct, or underproducing and excluding many valid transformations. Neither of these choices is acceptable.

1.4 Adding weights

While the development of a finite-state transducer toolkit was very helpful for attacking the problems of its age, advances in computation allowed modeling theory to expand beyond a level conceivable to the previous generation. Specifically, the availability of large corpora and the ability to process these corpora in reasonable time coupled with a move away from prescriptivist approaches to linguistics motivated a desire to represent *uncertainty* in processing and to empirically determine the likelihood of a processing decision based on these large corpora. Transducers that make a simple yes-or-no decision were no longer sufficient to represent models that included confidence scores. Researchers at AT&T designed FSM, a toolkit that harnesses *weighted* finite-state transducers—a superset of the formalism supported by XFST (Mohri, Pereira, and Riley, 2000). The association of weights with transducer rules affected the previous algorithms for composition and introduced new challenges. Along with the physical toolkit code, new algorithms were developed to cope with these challenges (Pereira and Riley, 1997; Mohri and Riley, 2002; Mohri, 2002). Carmel (Graehl, 1997), a toolkit with similar properties as FSM, but with an algorithm for EM training of weighted finite-state transducers (Eisner, 2002), was also useful in this regard. These toolkits and others like them were quite helpful for the community, as the state of NLP models had greatly expanded to include probabilistic models and without a decent weighted toolkit around, the only option to test these models was nose-to-the-grindstone coding of individual systems. Subsequent to their invention and release a number of published results featured the use of these toolkits and transducer formalisms in model design (Pereira, Riley, and Sproat, 1994; Sproat et al., 1996; Knight and Al-Onaizan, 1998; Clark, 2002; Zajic, Dorr, and Schwartz, 2002; Kumar and Byrne, 2003; Kolak, Byrne, and Resnik, 2003; Mathias and Byrne, 2006).

Returning to our translation example, we can see that some word sequences are more likely than others. Additionally, some translational correspondences are more likely than others. And perhaps we want to encourage the word reordering model to preserve English word order whenever possible. This information can be encoded using weights on the various rules. Consider the language model: both "green ball" and "green horse" are valid noun phrases, but the former is more likely than the latter. In the absence of any other evidence, we would expect to see "ball" after "green" more often than we would see "horse". However, we would expect to see "green horse" more often than "green the". By looking at a large amount of real English text, we can collect statistics on how often various words follow "green", and then calculate a probability for each possible word. Such probabilities are added to the relevant rules as follows:

- $q_{green} \xrightarrow{\text{ball/0.8}} q_{ball}$
- $q_{green} \xrightarrow{\text{horse/0.19}} q_{horse}$
- $q_{green} \xrightarrow{\text{the/0.01}} q_{the}$

The product of the probabilities of each rule used for a given sentence is the probability of the whole sentence. Notice that we did not exclude the very unlikely sequence "green the". However, we gave that rule very low probability, so any sentence with that sequence would be quite unlikely. This represents an increase in power over the unweighted model. Previously, we would not want to allow such a phrase, as it almost certainly would be wrong. Now we can still acknowledge the extreme unlikeliness of the phrase, but still allow for the rare situation where it is still the most likely choice.

We demonstrate our preferences in the other transducers in the chain through weights similarly. The reordering transducer, for instance, should favor some reorderings and disfavor others. An English noun phrase that contains an adjective (such as "green ball") would typically be translated with the noun first in Spanish. However, no reordering would be done for a noun phrase without adjectives (such as "the ball").

- $q \xrightarrow{\text{the:}\epsilon/0.1} q_{the}$
- $q \xrightarrow{\text{the:the/0.9}} q$
- $q_{the} \xrightarrow{\text{ball:ball the/1}} q$
- $q_{the} \xrightarrow{\text{green:green the/1}} q$
- $q_{the} \xrightarrow{\text{the:the the/1}} q$
- $q \xrightarrow{\text{green:}\epsilon/0.7} q_{green}$

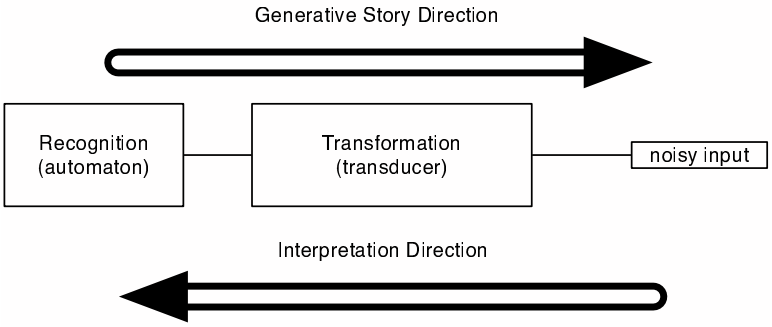


Figure 1.1: The general noisy channel model. The model is proposed in the "story" direction but used in the "decoding" direction, where a noisy input is transformed into the target domain and then validated against the recognizer.

- $q \xrightarrow{\text{green:green/0.3}} q$
- $q_{green} \xrightarrow{\text{ball:ball green/1}} q$
- $q_{green} \xrightarrow{\text{green:green green/1}} q$
- $q_{green} \xrightarrow{\text{the:the green/1}} q$

The translation model we have built is getting more and more powerful. It has begun to take on the shape of a very useful and often-applied general model of transformation—the *noisy channel model* (Shannon, 1948). The key principle behind this model, depicted in Figure 1.1, is the separation of a sensible transformation task into the cascade of a transformation task (without regard to sensibility of the output) followed by a recognition task that only permits sensible output. By simply substituting the appropriate transducer or transducers into our chain we can perform diverse tasks without altering the underlying machinery. If we remove the permutation and translation transducers from our model, and instead add a word-to-phoneme transducer followed by a phoneme-to-speech signal transducer, we can perform speech recognition on some given speech signal input. If we replace the word language model with a part-of-speech language model and the transducer cascade with a tag-to-word transducer we can perform part-of-speech tagging on some word sequence input. In all of these cases the only work required to transform our translation machine into a speech signal-recognition machine or a grammar markup machine is that of rule set construction, which is in its essence pure model design. The underlying mechanics of operation remain constant. This illustrates the wide power of weighted finite-state toolkits and explains why they have been so useful in many research projects.

1.5 Going further

We must acknowledge that our model is still a simplification of "real" human translation, and, for the foreseeable future, this will continue to be the case, as we are limited by practical elements, such as available computational power and data. This has long been a concern of model builders, and thus in every generation compromises are made. In the 1950s severe memory limitations precluded anything so general as a finite-state-machine toolkit. In the 1980s few useful corpora existed and normally available computational power was still too limited to support the millions of words necessary to adequately train a weighted transducer sequence. In the present age we have more computational power, and large available corpora, but much modeling of complex interaction such as translation is still done in a linear manner. We should consider whether there are yet more limitations imposed by the technology of the previous era that can now be relaxed.

One particular deficiency that arises, particularly in our translation model, is the requirement of linear structure, that is, a sequential left-to-right processing of input. Human language translation often involves massive movement depending on syntactic structure. In the previous examples we were translating between English and Spanish, two predominately Subject-Verb-Object (SVO) languages, but what if we were

translating between English and Japanese? The former is SVO but the latter is SOV. The movement of an arbitrarily long object phrase after an arbitrarily long verb phrase (or vice-versa) is simply not feasible with a formalism that at its fundamental core must process its input in linear order.

The disconnect between linear processes and the more hierarchical nature of natural language is an issue that has long been raised by linguists (Chomsky, 1956). However, practical considerations and a realization that a limited model with sufficient data was good enough for the time being led empirical research away from syntactic models for nearly half a century.

A survey of recent work in NLP shows there is evidence that syntax-rich empirical models may now be practical. Recent work with successful practical results in language modeling (Charniak, 2001), syntactic parsing (Collins, 1997), summarization (Knight and Marcu, 2002), question answering (Echihabi and Marcu, 2003), and machine translation (Yamada and Knight, 2001; Galley et al., 2004; DeNeefe et al., 2007), to name a few, clearly indicates there are gains to be made in syntactic NLP research. One common thread these papers have, however, is that the results behind the presented models were obtained via custom construction of one-off systems. Weighing the benefits of the syntax translation models of (Yamada and Knight, 2001) and (Galley et al., 2004), for example, requires access to the projects' respective code bases or re-engineering efforts. Consequently, few if any such comparative studies exist, and the cycle of model improvement is generally only possible for the original model engineers. Such a limitation is harmful to the community.

1.6 Better modeling through tree transducers

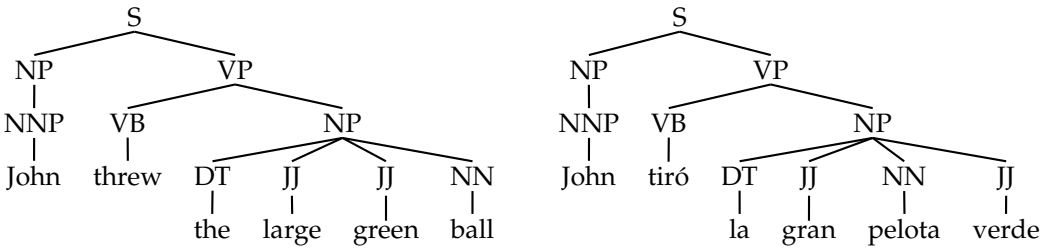
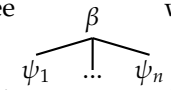
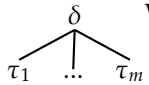


Figure 1.2: An (English, Spanish) tree pair whose transformation we can capture via tree transducers

The good news is many of these models can be expressed as a cascade of *finite-state tree transducers*. Tree transducers were invented independently by Rounds (1970) and Thatcher (1973) to support Chomskyan linguistic theories (Chomsky, 1957). However, it was the automata theory community that conducted extensive study into their properties (Gécseg and Steinby, 1984; Comon et al., 2007). Recently, weighted tree automata have been studied as a formal discipline as well (Engelfriet, Fülőp, and Vogler, 2001).

Let us consider how syntax can improve our previous toy translation model. Imagine we are now trying to translate between English and Spanish sentences annotated with syntactic trees, such as in Figure 1.2. We can accomplish this with top-down tree transducers, whose syntax we now informally describe. Whereas for our previous finite-state transducers (now referred to as finite-state *string* transducers to differentiate) we had rules of the form $\alpha \xrightarrow{\beta:\delta} \gamma$, here we may read in a portion of a tree  where β is a non-terminal node with n children and each ψ_i is a placeholder or *variable* denoting an unspecified but present subtree. Rather than simply writing an arbitrary length string δ and proceeding at some state γ, we may output a tree  where δ is a non-terminal node with m children and each τ_i is a subtree. A leaf of any τ_i may be a terminal node or it may be a pair (γ_j, ψ_j) where γ_j is some state and ψ_j is a variable from the read-in tree. This indicates that subsequent rules starting with γ_j and reading the tree represented by ψ_j should write their output at this point in the output tree. Because the input and output sequences are more

complex, we move them off of the transition arrow and write as α. $\beta \xrightarrow{\omega} \delta$ (with β over $\psi_1 \ldots \psi_n$ and δ over $\tau_1 \ldots \tau_m$), showing the output states in their appropriate places in the leaves of the tree headed by δ.

Since we are now considering syntax, we are no longer simply transforming between surface strings in English and Spanish, but between syntactic trees. What new power can our transducer chain have once we introduce this formalism?

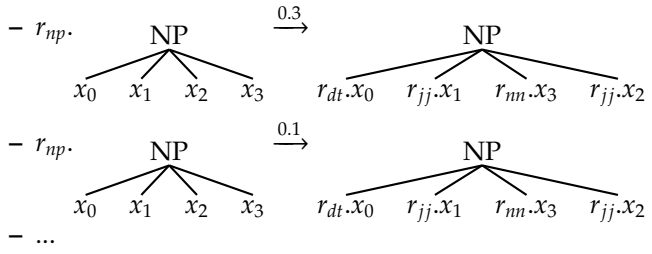
- Rather than recognizing valid English phrases, our new automaton A', the descendant of the string-based A, must now recognize valid English trees. Here's what some rules from A' could look like:

  - q
  - $q \xrightarrow{1}$ S ($q_{np-sing}$ $q_{vp-sing}$)
  - $q_{np-sing} \xrightarrow{0.7}$ NP (q_{dt} q_{nn})
  - $q_{np-sing} \xrightarrow{0.2}$ NP (q_{dt} q_{jj} q_{nn})
  - $q_{np-sing} \xrightarrow{0.1}$ NP (q_{dt} q_{jj} q_{jj} q_{nn})
  - $q_{dt} \xrightarrow{0.8}$ DT — the
  - $q_{dt} \xrightarrow{0.2}$ DT — a
  - $q_{jj} \xrightarrow{0.8}$ JJ — green
  - $q_{jj} \xrightarrow{0.2}$ JJ — large
  - $q_{jj} \xrightarrow{0.1}$ JJ — blue
  - $q_{nn} \xrightarrow{0.7}$ NN — ball
  - ...

  Already we can see some ways in which this syntactic automaton is more powerful than the string-based automaton. Words are conditioned on their parts of speech, so a more appropriate distribution for particular word classes can be defined. The top-down approach enables long-distance dependencies and global requirements. For example, the rule $q \rightarrow$ S ($q_{np-sing}$ $q_{vp-sing}$) indicates the sentence will have both a noun phrase and verb phrase, and that both will be singular, even though it is not yet known where the singular noun and verb will appear within those phrases.

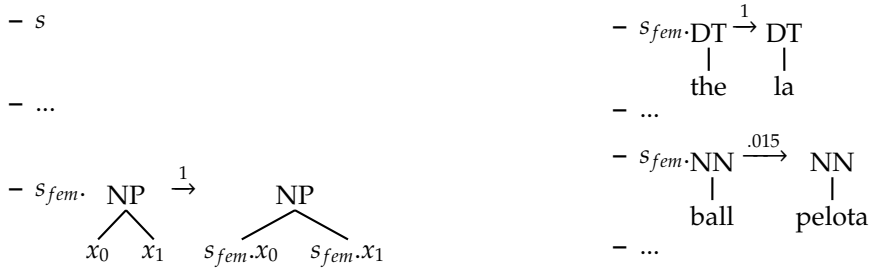
- Although we can now get quite creative with our permutation model, we can demonstrate the increased power of tree transducers by designing a B' that has the same idea expressed in B: allow re-ordering one level at a time:

  - r
  - $r.$ S(x_0 x_1) $\xrightarrow{0.7}$ S($r_{np}.x_0$ $r_{vp}.x_1$)
  - $r.$ S(x_0 x_1) $\xrightarrow{0.3}$ S($r_{vp}.x_1$ $r_{np}.x_0$)
  - $r_{np}.$ NP(x_0 x_1 x_2 x_3) $\xrightarrow{0.4}$ NP($r_{dt}.x_0$ $r_{nn}.x_3$ $r_{jj}.x_1$ $r_{jj}.x_2$)



The reordering at the top of the tree, swapping the positions of arbitrarily large noun phrases and verb phrases, does sentence-wide permutation in a single step. Doing the equivalent with string-based transducers would require a unique state for every possible noun phrase, making such an operation generally impossible. The lower-level inversion could feasibly be accomplished with a string transducer, but it would still require state that encodes the sequence of adjectives seen until the noun at the end of the phrase is reached.

- We can use the power of syntax to our advantage in the design of the translation transducer, C'. Recall that C has a set of word-to-word rules that do not take context into account. We can easily take context into account in C' by state:



State can also be used to encode number (differentiating "el" from "los") and case (e.g. "yo" vs "me").

1.7 Building a new toolkit

Weighted tree automata are a powerful formalism for natural language processing models, and as the example above indicates, constructing useful models is not difficult. However, there is one main roadblock preventing progress in weighted tree automata modeling of the scale seen for weighted string automata: No appropriate toolkit exists. There are some existing tree automata toolkits (Borovansky et al., 1996; Genet and Tong, 2001; Henriksen et al., 1995) but these are unweighted, a crucial omission in the age of data-driven modeling. Additionally, they are chiefly aimed at the logic and automata theory community and are not suited for the needs of the NLP community.

Although no toolkit currently exists, it is a simple matter to look at the history of toolkit development and the relationship between the structures we wish to support and those that were handled in weaker toolkits and infer what such development entails. We should of course follow the lead of other toolkits in having simple design semantics and a relatively simple and small set of operators. We should be able to read and write large text files that contain training data and transducers, expressed as rules. We will need to support operations such as composition, projection, and training. Algorithms for these operations as they relate to weighted tree transducers may not be known. We will thus have to discover or invent new algorithms as necessary, and verify their correctness.

It is not enough to simply provide a toolkit for a different type of finite-state machine and assume such a toolkit is usable. Our running example notwithstanding, we should justify the formalism we are supporting by providing evidence of its applicability to a wide range of current work and demonstrate how it may be used in future work. To that end we must investigate existing syntax-based models and describe their formulation as tree transducer cascades. We must then show the toolkit is up to the challenge of replicating the work previously done on custom systems. We can then describe new work, fully imagined

in the language of these tree transducers, and demonstrate the use of the toolkit to attack real problems and achieve real empirical gains.

With that in mind, these are the contributions I will provide in my thesis:

- I propose to write Tiburon, a tree transducer toolkit that provides fundamental operations such that complicated tree-based models may easily be represented. The toolkit will provide implementations of useful algorithms and a simple interface such that high-quality research projects and experiments may be carried out with a minimum of custom programming work.
- I will provide several novel algorithms that accomplish for practical weighted tree automata theory what the algorithms of (Mohri, Pereira, and Riley, 2000) accomplished for string automata. These new algorithms for composition, determinization, minimization, and semantics generation support the structures of Tiburon in a manner analogous to the relationship between the algorithms in (Mohri, Pereira, and Riley, 2000) and the AT&T toolkit.
- I will demonstrate the power and limitation of tree automata by a detailed treatment of the translation model of (Yamada and Knight, 2001) and the parsing model of (Collins, 1997) in the language of tree automata. These treatments allow direct reimplementation of those complex models within Tiburon and offer insight into the nature of syntactic models.
- I will describe new models for machine translation and parsing using combinations and manipulations of tree automata and demonstrate empirical performance gains over state of the art performance on large-scale tasks. The systems built to accomplish these tasks use Tiburon and have clean, uniform descriptions that cannot hide implementation details because the underlying machinery is entirely generic.

Chapter 2

A Brief Review of Finite-State Machines and Algorithms

The machine unmakes the man.

Ralph Waldo Emerson

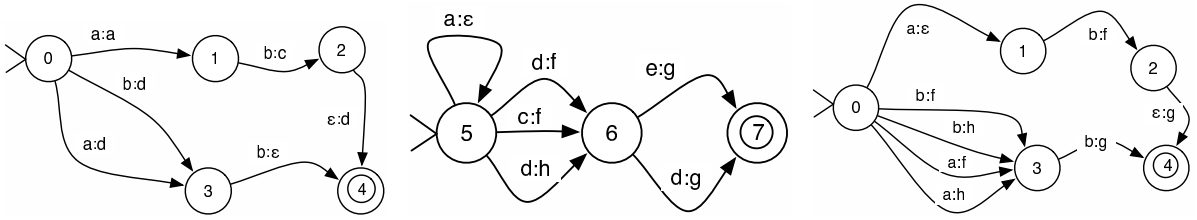
In Chapter 1 we characterized finite-state transducers and automata as having useful properties that allowed a wide range of modeling to be performed with a core set of algorithms. In this chapter we discuss those properties and describe the algorithms, explaining just how they are useful. As we expect the reader to be somewhat familiar with finite-state string transducers and automata, and less so with their tree counterparts, we present finite-state weighted tree transducers and automata as a dual generalization of the more well-known formalisms and work our way up the complexity chain, explaining algorithms once they become relevant. Eventually we reach weighted tree automata and extended weighted tree transducers, the formalisms behind future chapters' models and Tiburon's main formalism.

A note about terminology. So that we can avoid the cumbersome "finite-state transducers and automata" when talking about both together, in this work we refer to the broad class of finite-state systems as *finite-state machines*. These can be roughly divided into *transducers*, machines that both read and write, and *automata*, machines that either read or write[1]. For most of our purposes an *acceptor*, or reading automaton, is indistinguishable from a *generator*, or writing automaton; where such a distinction is necessary we will make it. In much of the literature string input and/or output is assumed, thus works dealing with, for example, finite-state *string* transducers drop the distinction and refer instead to *finite-state transducers*. Since we are concerned with relationships between finite-state tree and string machines, we will emphasize the differences, referring to, for example *tree transducers* and *string transducers*, and generally omit the "finite-state" designation, unless clarification is needed.

2.1 String machines

Figures 2.1(a), 2.1(b), and 2.1(c) are visualizations of *string transducers*. String transducers have a set of *states*, one of which is a start state and several of which are final states, and a set of *rules* which dictate a symbol read, a symbol written, and the state transition that accompanies this processing. In the graphical forms depicted in Figure 2.1 and subsequently throughout this work, states are indicated by circles, labeled for convenience of reference. A bodyless arrow points to the start state, and a double circle indicates a final state. Rules are indicated by directed arrows between states and are annotated with the symbols read and/or written. The special symbol ϵ appearing in either the input or output position of a rule indicates no word is read and/or written during the state transition. Equivalent grammar representations of the transducers are in

[1] There is confusion in the literature; some sources use *automata* in the sense we use *machines* (e.g., (Gécseg and Steinby, 1984)) and some use it as we do (e.g., (Borchardt and Vogler, 2003)).



(a) String transducer A (b) String transducer B (c) Composition of A and B

- start = 0
- final = 4
- $0 \xrightarrow{a:a} 1$
- $0 \xrightarrow{b:d} 3$
- $0 \xrightarrow{a:d} 3$
- $1 \xrightarrow{b:c} 2$
- $2 \xrightarrow{\epsilon:d} 4$
- $3 \xrightarrow{b:e} 4$

(d) Grammar for transducer A

- start = 5
- final = 7
- $5 \xrightarrow{a:\epsilon} 5$
- $5 \xrightarrow{d:f} 6$
- $5 \xrightarrow{c:f} 6$
- $5 \xrightarrow{d:h} 6$
- $6 \xrightarrow{e:g} 7$
- $6 \xrightarrow{d:g} 7$

(e) Grammar for transducer B

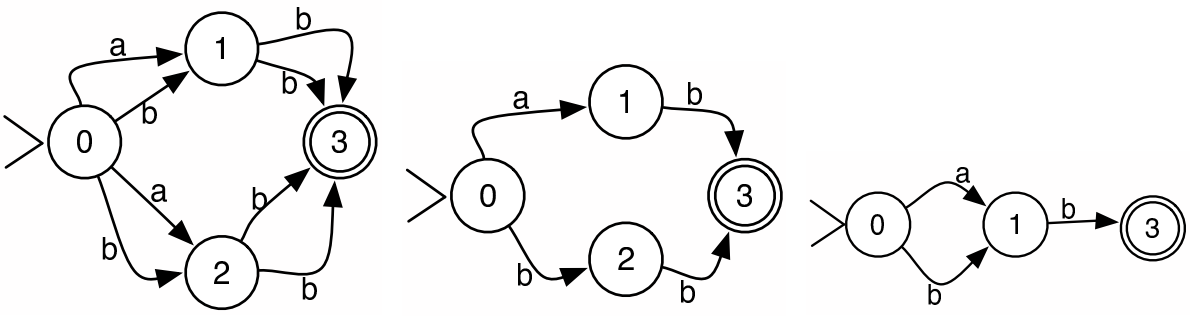
- start = 0
- final = 4
- $0 \xrightarrow{a:\epsilon} 1$
- $1 \xrightarrow{b:f} 2$
- $2 \xrightarrow{\epsilon:g} 4$
- $0 \xrightarrow{b:f} 3$
- $0 \xrightarrow{b:h} 3$
- $0 \xrightarrow{a:f} 3$
- $0 \xrightarrow{a:h} 3$
- $3 \xrightarrow{b:g} 4$

(f) Grammar for composition

Figure 2.1: Two string transducers and their composition

Figures 2.1(d), 2.1(e), and 2.1(f), respectively. String transducers are *closed under composition* (Schützenberger, 1961). This means a single string transducer can be built that does the work of any two sequential transducers. Figure 2.1(c) is formed from the composition of the transducers of Figures 2.1(a) and 2.1(b). The corresponding algorithm that implements this construction naïvely runs in $O(|M_1||M_2|)$ time where $|M_x|$ is the size (in the number of rules and states) of transducer M_x; careful construction can lessen that in practice, particularly when more than two transducers are composed (Allauzen and Mohri, 2007). The composition algorithm is the "glue" that chains a transducer cascade together and is thus useful in automatically combining two separately-behaving transducers.

Figure 2.2 shows three examples of *string automata*. Automata are similar to transducers except that they are only used to recognize or generate sentences and thus each rule has a single symbol label. As string automata are equivalent to string transducers where each rule has the same input and output label, the composition algorithm for transducers can be used to find the *intersection* of two automata. Notice there are multiple different paths in Figure 2.2(a) that read the string "a b". An automaton with more than one path for a given accepted string is known as a *non-deterministic* automaton. Deterministic automata are easier to check for membership than non-deterministic automata because no backtracking is required; if the path explored does not result in reaching a final state when reading some input, no path will. However, non-deterministic automata are often easier to design and can be specified more compactly than their deterministic counterparts. Thankfully, it is known that both types of string automata recognize the same language family, and a *determinization algorithm* exists to transform a non-deterministic string automaton into a deterministic automaton that recognizes the same language (Rabin and Scott, 1959). Figure 2.2(b) shows the result of determinizing Figure 2.2(a). One potential consequence of determinization, however, is a blow-up in the number of states in the resulting automaton. To mitigate this, as well as to shrink automata constructed by other means, it is helpful to obtain determinized automata that recognize the same language as some given automaton but have fewer rules and/or states. Thankfully there exists a unique minimal



(a) Before determinization (b) After determinization (c) After minimization

Figure 2.2: A string automaton in non-determinized, determinized, and minimized form

deterministic string automaton for any regular language, as well as an algorithm to convert a deterministic string automaton into its equivalent minimal form (Aho, Hopcroft, and Ullman, 1974). Figure 2.2(c) shows the result of minimizing Figure 2.2(b).

2.2 Weighted string machines

Practical results in finite-state machine work have benefited greatly from the idea of associating confidence metrics with valid paths that is accomplished by the elevation to *weighted string machines*. The formalism for weighted machines augments that for the unweighted case by associating weights to rules and defining a method for combining weights when reading along a single path and when combining alternate paths. As in the unweighted case, we would like to have the ability to compose weighted string transducers in order to create transducer cascades. Like their unweighted counterparts weighted string transducers are closed under composition, though care needs to be taken to ensure weights are preserved properly. (Pereira and Riley, 1997) presents an algorithm that correctly composes weighted string transducers. Naturally, the same algorithm can be used for the intersection of weighted string automata. Recently, more sophisticated versions of this algorithm have been presented that are particularly suited for composition of more than two transducers in a chain (Allauzen and Mohri, 2007).

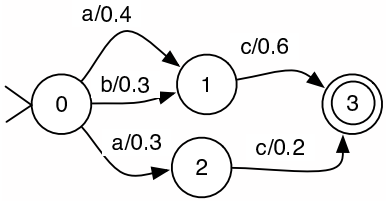


Figure 2.3: A weighted string automaton

Figure 2.3 is an example of a weighted string automaton. In this example the multiplication operator is used to combine weights along a path and the addition operator is used to combine alternate paths. Thus, the weight for string "b c" is 0.18 and the combined weight for both paths reading string "a c" is 0.3.

Assigning weights to the paths invites us to inquire about the highest weighted path in a weighted machine. We can further generalize to inquire about the k highest weighted paths. There are numerous algorithms to obtain this; the fastest known (Eppstein, 1998) has worst-case runtime $O(m + n \log n + k)$ for an automaton with m rules and n states. Aside from the k highest weighted paths we may also want the k highest weighted *strings* in an automaton or string pairs in a transducer, i.e., we want our k-best list to have no repeated members. A determinization algorithm for weighted string automata that ensures weights in the single path for a given string or string pair has the sum of the weights in the multiple paths of the string or pair (Mohri, 1997; Allauzen and Mohri, 2004) prior to obtaining the k-best paths accomplishes

(a) Noun phrase as tree (b) String as special case of tree

Figure 2.4: Two trees

this goal. The previously cited advantages from determinization are also gained by this algorithm. As determinization can increase the size of the input exponentially, minimization algorithms are also desired, and these have been found for certain deterministic weighted string automata (Mohri, 1997; Eisner, 2003). Minimization of transducers has in general not been studied to the best of our knowledge.

Even with the best efforts of minimization, it is still often desirable to reduce weighted string machines in a *lossy* way, sacrificing the represented language in the name of practical considerations such as time and space. This is accomplished by *pruning* some rules from the machine. As we are generally concerned with the most likely strings or pairs in a language, we would like to prune off just those rules involved in the least likely paths. Although not specifically described in the literature, An $O(m+n)$ algorithm to do this has been implemented in at least two toolkits (Graehl, 1997; Riley et al., 2007).

We have not yet described where these weights come from. Naturally, they can be set by hand by experts who carefully craft the rules and have knowledge of the particular domain in which the machine will operate. However, many model constructors are interested in building models that reflect the statistics of data seen in practice. A *training algorithm* that, given training data, assigns probability to rules in such a way that the probability of the training data is maximized is highly useful. The EM algorithm (Dempster, Laird, and Rubin, 1977) is appropriate for this task. An instantiation of the algorithm for transducers, which is also applicable to automata, is given in (Eisner, 2002) and the Carmel Finite-State toolkit (Graehl, 1997) provides an implementation.

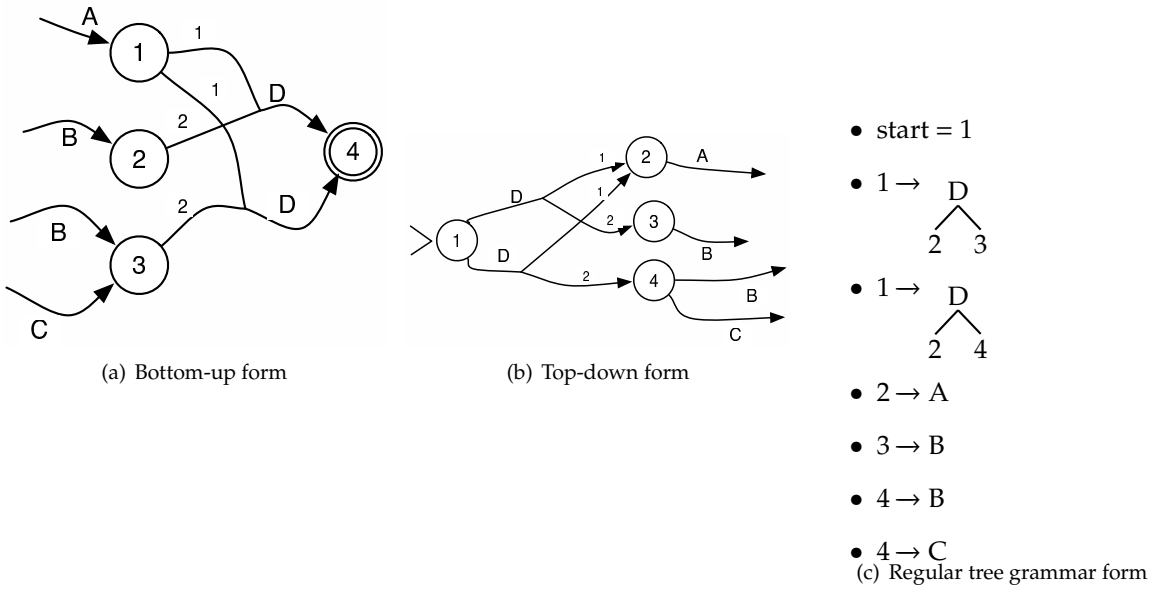
2.3 Tree machines and weighted tree machines

The literature further generalizes finite-state machines to include processing of *trees*. A tree is a symbol with zero or more ordered children, each of which is also a tree. Thus, a string is a special case of a tree, as a string may be defined as a symbol with at most one child. A tree with zero children is known as a *leaf*, particularly if it is the child of some tree. Figure 2.4 has two example trees. Figure 2.4(a) is a typical example of the sort of tree we will be concerned with in this work. In this case the symbol "NP" has rank 2, the symbols "DT" and "NN" have rank 1, and the *frontier* symbols, those with rank 0, are "the" and "tree". Figure 2.4(b) is an example of the special case of strings represented as trees. Since all words in a string should have the same rank in their tree notation we introduce the special "END" symbol with rank 0. All meaningful words in this representation have rank 1.

Figure 2.5 is an example of a tree automaton. Like string automata, tree automata contain states and rules dictating state transitions and reading and writing of symbols. The principal difference is that the symbols read and written are pieces of a tree and thus have multiple children. This is why Figures 2.5(a) and 2.5(b) are *hypergraphs*. This automaton is presented in bottom-up form in Figure 2.5(a), and in top-down form in Figure 2.5(b). Note that bottom-up tree automata have no explicit start state, as rules with no source state are allowed at the beginning. Similarly, top-down tree automata have no explicit final state, and processing is done when symbols with no children are read.

Tree automata are fairly difficult to read in graphical form, so the equivalent textual *regular tree grammar* (RTG) formalism is often used instead. An equivalent RTG to the automaton of Figure 2.5(b) is presented in Figure 2.5(c). RTG are also applicable to automata read bottom-up; in this case there is no explicit start state and the denoted state is a terminating state.

Deterministic top-down tree automata, defined analogously to their string counterparts, are strictly weaker than non-deterministic top-down tree automata. However, non-deterministic and deterministic bottom-up tree automata are equivalent (and are equivalent to top-down non-deterministic tree automata as well), so a straightforward generalization of the technique used for string automata exists for bottom-



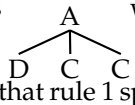
(a) Bottom-up form

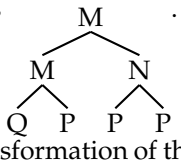
(b) Top-down form

(c) Regular tree grammar form

Figure 2.5: A tree automaton in three equivalent forms.

up determinization (Doner, 1970). Algorithms to minimize a determinized tree automaton are found in (Brainerd, 1968) and (Arbib and Give'on, 1968).

As might be expected from the name, tree transducers transform trees in much the same way string transducers transform strings. Most literature uses the term "tree transducer" to mean a machine that both reads in and writes out trees. We use that term to describe any transducer that reads a tree and draw a distinction between *tree-to-tree* and *tree-to-string* transducers, where the former is the classic type of tree transducer and the latter reads a tree but writes a string. Figure 2.6(a) depicts a tree-to-tree transducer in grammar form[2] and Figure 2.6(b) a similar tree-to-string transducer. A prime difference between tree transducer rules and string transducer rules (other than the presence of trees) is the incorporation of variables into the leaves of the rules' input and output. This is done to specify the location in the output tree or string at which continued writing takes place for each yet-unread input subtree. As an example, consider the rules in Figure 2.6(a). If the input to this transducer is 

we apply rule 1, then, in order, rules 6, 3, and 5, and the transducer outputs 

. Note that rule 1 specifies the location of processing of subsequent subtrees, such that the transformation of the right-most leaf of the input tree, C, is reflected on the left of the output tree as Q, and so forth.

Tree transducers are considered *copying* if they process input more than once; rule 2 in Figure 2.6(a) is an example of a copying rule. They are considered *deleting* if they do not process some of their input; rule 7 is an example of a deleting rule. Most tree transducers discussed in the literature constrain their rules such that at most a single symbol and set of immediate child variables is read per rule. We will at times discuss a class of *extended* tree transducers that does not have this constraint; rule 8 is an example of such a rule.

As with string transducers we would like to be able to form a chain of tree-to-tree transducers, possibly with a tree-to-string transducer on the end, and compose them together to form a single tree transducer. However, tree transducers with copying and deleting power are not closed under composition (Baker, 1979; Gécseg and Steinby, 1984). If extended tree transducers are considered, closure under composition is not guaranteed even if copying and deleting are forbidden (Maletti et al., 2008). For cases where closure under

[2]There is no suitable graphical visualization for tree transducers so we will only present them as grammars

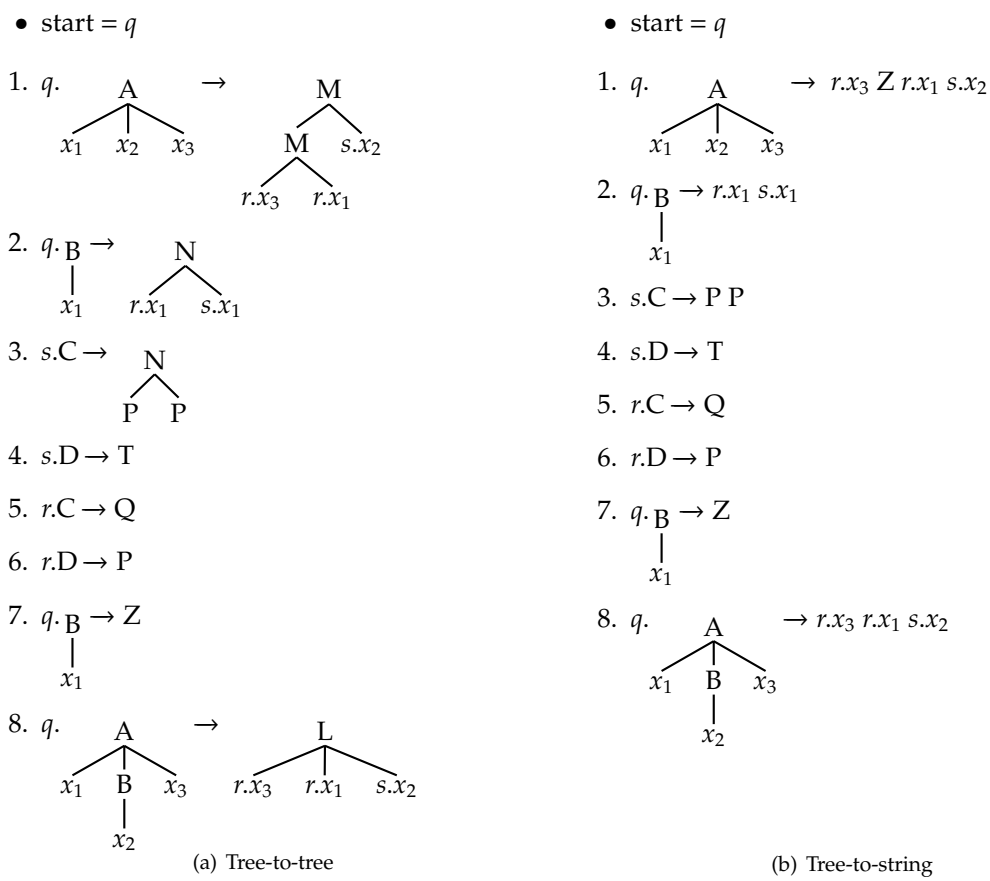


Figure 2.6: Two top-down tree transducers

composition is guaranteed a suitable algorithm to perform it is desired. Where composition is not possible but the effect of composition is, by iteratively processing an input through a series of transducers, we also would like to have suitable algorithms. We know of no such explicit algorithms available in the literature, and present several as novel contributions, detailed in Chapter 3. These techniques are also useful in obtaining the intersection of two tree automata.

Projection, obtaining the input or output automaton that represents the language a transducer can read or write, is a trivial operation for string transducers, as this amounts to ignoring the undesired side of transduction. While projection is often trivial for tree transducers, depending on their property and the side of projection that is desired, this is not always true. Note that the range projection of a tree-to-string transducer, if possible, will be a context-free grammar (CFG). In Chapter 5 we discuss plans to present useful algorithms for projection, which are also useful in composition and application.

Tree machines can be extended with weights in the same way as string machines. With weighted tree machines we can consider weight-related algorithms and extensions to algorithms for unweighted tree automata.

An algorithm for obtaining the k most likely paths in a weighted tree automaton with m rules and n states that runs in $O(m + nk \log k)$ time is presented in (Huang and Chiang, 2005). Determinization for weighted tree automata, which allows the k-best algorithm to obtain a list of best trees instead of best paths, was discussed in (Borchardt and Vogler, 2003), but the presented algorithm creates a state for each possible path. In Chapter 3 we present a more efficient algorithm, discussed in (May and Knight, 2006a), that extends the weighted string algorithm of (Mohri, 1997) to the case of weighted tree automata without loops. A recent algorithm for minimizing deterministic weighted tree automata has been proposed in (Maletti, 2008). The desire for lossy minimization, that is, pruning, is also present for tree machines, and we can apply the same principles for string machines in an appropriate algorithm.

Algorithm	String		Tree	
	unweighted	weighted	unweighted	weighted
determinization	Yes	Yes	PoC	No
minimization	Yes	Yes	Alg	No
bisimulation minimization	Alg	Alg	Alg	No
K-best	N/A	Yes	N/A	Alg
pruning	N/A	Yes	N/A	Alg
intersection	Yes	Yes	PoC	PoC
EM training	Yes		Alg	

Table 2.1: Available implementations of algorithms for various classes of automata. Yes = an algorithm is known and an implementation is publicly available. Alg = an algorithm is known but no implementation is known to be available. PoC = a proof of concept of the viability of an algorithm is known but there is no explicit algorithm. No = no methods have been described

Algorithm	String		Tree	
	unweighted	weighted	unweighted	weighted
composition	Yes	Yes	PoC	PoC
domain and range projection	Yes	Yes	PoC	PoC
application	Yes	Yes	PoC	PoC
EM training	Yes		Alg	

Table 2.2: Available implementations of algorithms for various classes of transducers, using the key described in Table 2.1

For the purposes of compact representation, it may be useful to obtain minimal *non-deterministic* weighted tree automata. There is not guaranteed to be a single minimal non-deterministic weighted tree automaton for some given weighted tree automaton and even if there is one, an algorithm for finding it is PSPACE-complete (Meyer and Stockmeyer, 1972). In Chapter 3 we discuss a new algorithm that performs a heuristic minimization approximation for non-deterministic weighted tree automata (Högberg, Maletti, and May, 2007c).

Naturally, the difficulties associated with tree transducer composition are present as well with weighted tree transducer composition. Additionally, we must take into consideration the ambiguities weights bring to composition algorithms, that were handled in the string case by filters in (Pereira and Riley, 1997). To our knowledge there has been no serious study of this issue.

Training of weighted tree machines, as in the string case, is a useful way of setting weights on tree machines that reflect a training corpus of real-world data. An EM training procedure for weighted tree transducers is presented in (Knight, Graehl, and May, 2008) and is trivially adaptable for automata.

The algorithms discussed in this chapter and their existence as published algorithms or available implemented code prior to this work are outlined in Tables 2.1 and 2.2. In Chapter 3 we discuss our contribution toward changing those items in the chart labeled "No" to "Yes" and in Chapter 4 we discuss the use of our implementations of some of the elements labeled "Alg". Chapter 5 details plans to turn all appropriate cells in Tables 2.1 and 2.2 to "Yes".

Chapter 3

Algorithmic Contributions

> It is safer to accept any chance that offers itself, and extemporize a procedure to fit it, than to get a good plan matured, and wait for a chance of using it.
>
> Thomas Hardy

In this chapter we describe our contributions to the body of algorithms outlined in Chapter 2. Specifically, we describe an algorithm for (bottom-up) determinization of weighted tree automata and a heuristic minimization algorithm for weighted tree automata that is applicable to non-deterministic and deterministic variants.

3.1 Practical weighted determinization of tree automata

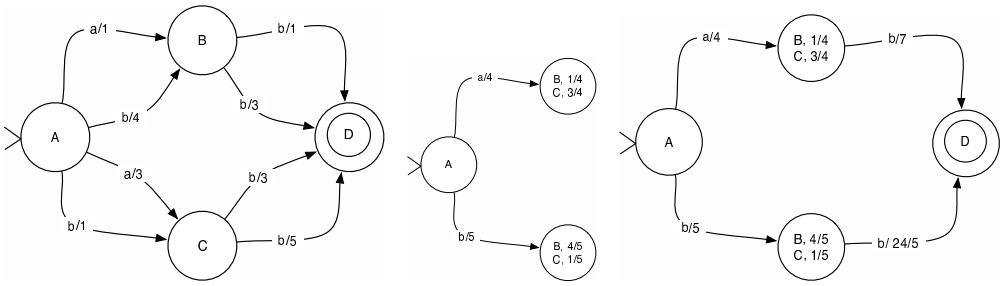
In (May and Knight, 2006a) we presented an algorithm for determinizing a weighted tree automaton for the purpose of generating *k*-best lists of trees in the automaton's language, rather than *k*-best lists of paths, where the same tree could be recognized in multiple paths. There is a natural empirical motivation for such an algorithm in natural language. Ranked lists of output trees are useful for re-ranking and tuning algorithms that can improve systems that parse, recognize speech, and automatically translate, to name a few. However the systems suffer if the lists are highly repetitive, and the nature of system construction, where multiple partial results are combined heuristically, ensures that they are.

As proposed in Chapter 2, we also value obtaining deterministic weighted tree automata so that checking a tree's membership does not require backtracking. Both of these problems are solved by applying a determinization algorithm to an input weighted tree automaton.

3.1.1 Mohri algorithm

The motivation for our algorithm is taken directly from (Mohri, 1997), which presented an algorithm for determinization of weighted string automata. The algorithm combines subset construction with a bookkeeping system that keeps track of leftover weight produced as a result of removing path ambiguity. The algorithm is described in detail in (Mohri, 1997) but a motivating example will suffice to explain it here.

The string automaton in Figure 3.1(a) nondeterministically recognizes the strings "a b" and "b b" in a variety of ways, with a variety of weights (for this example we assume the weight of a path is determined by multiplying the weights of the rules along the paths and the weight of a string is determined by adding the weights of all paths that recognize the same string). We use the determinization algorithm of (Mohri, 1997) to obtain an automaton that contains one path for each recognized string, with weight equal to the sum of all paths in the input automaton. The method taken is a subset construction that keeps track of the



(a) Original FSA (b) Partway through determinization (c) After determinization

Figure 3.1: Determinization of FSA (modification of figure from (Mohri, 1997))

method	BLEU
undeterminized	21.87
top-500 "crunching"	23.33
determinized	24.17

(a) BLEU results from string-to-tree machine translation of 116 short Chinese sentences with no language model.

method	Recall	Precision	F-measure
undeterminized	80.23	80.18	80.20
top-500 "crunching"	80.48	80.29	80.39
determinized	81.09	79.72	80.40

(b) Recall, precision, and F-measure results on DOP-style parsing of section 23 of the Penn Treebank.

Table 3.1: Performance improvements using determinization of weighted tree automata. In each experiment results are shown for the best derivation (undeterminized), estimate of best tree (top-500), and true best tree (determinized)

weights of individual rules. As an example, consider the start state, A, which has 2 outgoing rules for the label "a". Our new automaton will have one rule labeled "a" coming from the start state, with weight equal to the sum of the non-deterministic "a"-labeled rules, i.e., 4. What state should this rule go to? A classic subset construction dictates it goes to a state representing both states B and C of the original automaton. But we wish to be able to recover the individual rule weights, so we associate numbers, called "residuals" in (Mohri, 1997), that allow this recovery. We associate 1/4 with B and 3/4 with C. By multiplying the appropriate residual with the incoming rule we can thus recover the original weights. This also indicates, in some sense, that 1/4 the total weight reached at this state is due to state B and 3/4 due to state C. The analogous operation for rule "b" is also performed, and the partially built deterministic automaton is shown in Figure 3.1(b).

To determine the outgoing rules of a state that represents a subset of states, we consider all the outgoing rules coming from the original states represented and the residual from the original state. For state "B, 1/4 C, 3/4", rules labeled "b" come from original state B, with a sum of 4, scaled by multiplying in the residual for B to 1. Rules coming from C have a sum of 8, scaled by multiplying in the residual for C to 6. The total weight, then, for the rule from this state labeled "b" is 7. All rules labeled "b" lead to state D, so the destination state is simple to determine. The analogous operations are done from the state "B, 1/5 C, 4/5" and the result determinized automaton is showed in Figure 3.1(c). The reader can verify the score for each string in the deterministic automaton is equal to the sum of the weights of all paths over each string in the original of Figure 3.1(a).

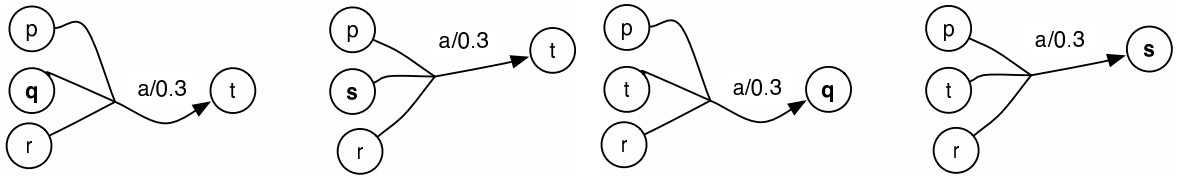
3.1.2 Our Algorithm

As described in (May and Knight, 2006a) we elevate the algorithm of (Mohri, 1997) to the bottom-up tree automata case. The essential differences from the previous algorithm are that a vector of source states replaces a single source state, and the residuals of all the states in a source vector are multiplied in when calculating weights and subsequent states. This algorithm proved useful on empirical experiments. Determinization of forests of output trees generated by a syntax-based machine translation system (Galley et

al., 2004) changed the tree regarded as "best" 77.6% of the time and raised BLEU scores considerably. When applied to a Data-Oriented Parsing-like model (Bod, 2003) we also showed empirical gains by determinizing parse forests and returning the best tree in the forest, rather than the best path in a non-deterministic forest. Table 3.1 compares performance in these tasks when the output forests are undeterminized, when a top tree is estimated by summing duplicates over the top 500 paths in the forest, and by using the true top tree via weighted determinization.

3.2 Bisimulation minimization of weighted tree automata

In (Högberg, Maletti, and May, 2007c) we presented the results of a collaborative effort on a weighted minimization algorithm for tree automata. In general a non-deterministic tree automaton may not have a single form with a minimal number of states, and in the case that it does finding such an automaton is PSPACE-complete (Meyer and Stockmeyer, 1972). However by using the principles of bisimulation, essentially a local heuristic that determines whether or not a subset of states can be merged or must remain separate, we produced a polynomial-time algorithm that empirically showed good performance when run on an automatically-generated dictionary of parsed English trees.



(a) Forward bisimularity (b) Backward bisimularity

Figure 3.2: Examples of bisimularity conditions for states q and s

3.2.1 Bisimulation

Two systems are *bisimular* if their behavior given an input is indistinguishable to outside observers. This concept is applied to minimization of tree automata by considering, pairwise, the states of an automaton and merging them if they are bisimular according to a local test. (Högberg, Maletti, and May, 2007a) and (Högberg, Maletti, and May, 2007c) present two such local tests for bisimularity as well as accompanying algorithms. The first, *forward bisimulation*, considers states to be bisimular if they only contain rules that ensure their futures are the same. Alternatively, *backward bisimulation* considers states bisimular if their pasts are the same. Figure 3.2 contains examples of bisimularity conditions for states q and s. Notice that this algorithm is applied to bottom-up tree automata. Thus, the notion of equivalent futures is, roughly, identical completion on top of several (possibly different) already-recognized subtrees. The notion of equivalent pasts is an identically completed sequence of already recognized-subtrees. Future bisimularity is identified by a local test as follows:

A *context* c is a sequence of states where one state has been replaced by the symbol $\square$. It is used to describe the states in the source of a rule other than the state in question. For example, in Figure 3.2(a) the rules depicted both have context $p \square r$. $c[[x]]$ denotes a context c with the $\square$ "filled" by state x. If c is $p \square r$ the rules of Figure 3.2(a) have source sequences $c[[q]]$ and $c[[s]]$, respectively. Two states p and q are locally future-bisimular with respect to some set of destination states B if $\sum_{r \in B} w_\sigma(c[[p]], r) = \sum_{r \in B} w_\sigma(c[[q]], r)$ for all recognizable symbols σ and contexts c, where $w_\sigma(\{x_1 ... x_n\}, y)$ is the weight of a rule $y \xrightarrow{w} \sigma(x_1, \ldots, x_n)$.

The local test for backward bisimularity is analogous to that for forward bisimularity, though naturally inverted. We extend the definition of w such that $w_\sigma(\{X_1 ... X_n\}, y)$ where X_i is some subset of states is the sum of all $w_\sigma(\{x_1 ... x_n\}, y)$ where $x_i \in X_i \forall i \in [1, n]$. Let R be a partition of the state set, represented as a set of sets of states, and let B be an identified member of R. $R^k(B)$ is a subset of R^k where each member contains at least one instance of B. Then, two states p and q are locally past-bisimular with respect to some partitioning

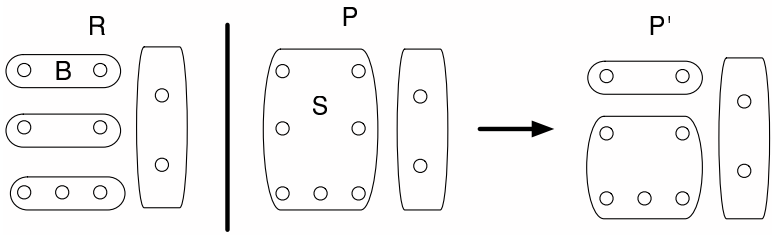


Figure 3.3: Demonstration of the refinement of P. A block of P is chosen as S, and a subset of S from R is chosen as B. P' differs from P in that it replaces S with B and $S \setminus B$

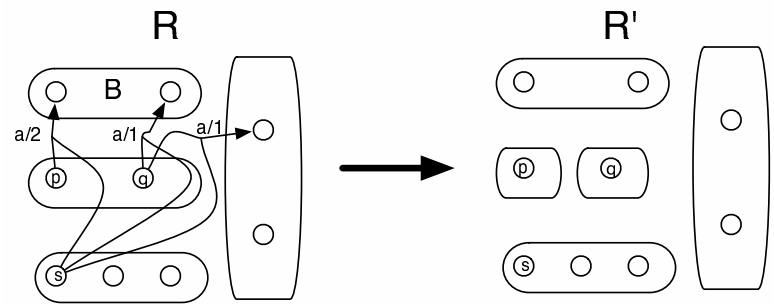


Figure 3.4: Demonstration of the refinement of R with the forward bisimulation algorithm. States p and q, while future-bisimular with respect to the entire state set, are not future-bisimular with respect to block B and are thus split.

R and some identified member B if $\sum_{\vec{X} \in R^k(B)} w_\sigma(\vec{X}, p) = \sum_{\vec{X} \in R^k(B)} w_\sigma(\vec{X}, q)$ for all recognizable symbols σ with rank k. These two local bisimularity tests are used in the algorithms for minimization.

3.2.2 Algorithms

The algorithms operate following the "process the smaller half" approach used by Hopcroft in the minimization of deterministic string automata (Hopcroft, 1971). We initially are optimistic and assume that all states may be considered equivalent and merged, then attempt to confirm our assumption by applying local tests to blocks of states, splitting the blocks apart if these tests fail. Once the algorithm terminates a new automaton consisting of one state for each block is returned.

Throughout the algorithm we maintain two partitions of states into equivalence classes. P, a "coarse" partition, and R, a refinement of P. Initially, P consists of a single block with all states and R has all states that are not future-bisimular with respect to the entire state set. In each iteration P and R are refined to P' and R', respectively, until they are the same, at which time the algorithm terminates. P' is formed from P by choosing a block S from P that is represented by at least two blocks in R, choosing B, one of those blocks in R, and splitting B from S in the new P. This is illustrated in Figure 3.3. A new R is formed by splitting any states in the same block that are not future-bisimular with respect to B. This is illustrated in Figure 3.4.

The backward algorithm is straightforward given the forward algorithm. The initial P is the same, and the initial R is split by states that are not past-bisimular with respect to the entire state set. The splitting of P in each iteration is the same as in the forward algorithm. The splitting of R is also the same, except that blocks of R are split with regard to states that are not past-bisimular with respect to P'.

3.2.3 Results

In (Högberg, Maletti, and May, 2007a; Högberg, Maletti, and May, 2007c) we applied forward and backward bisimulation to the problem of compactly representing partial syntax trees, which is useful for tree language modeling. We prepared a data set by collecting 3-subtrees, i.e., all subtrees of height 3, from sentences taken

	states	rules	states	rules	states	rules	states	rules
5	23	23	18	22	21	21	16	20
105	707	707	598	694	475	475	361	457
205	1366	1366	1130	1324	840	840	575	769
305	1996	1996	1630	1924	1143	1143	735	1029

Table 3.2: Reduction of states and rules by using the bisimulation minimization algorithms

from the Penn Treebank. An initial weighted tree automaton was constructed by representing each 3-subtree in a single path. We then applied both the forward and backward minimization algorithms to our initial automaton. Automata can be minimized by iteratively running the forward and backward algorithms in alternating succession until no changes are observed. However, although the choice of initial minimization can affect the end automaton, we found the choice of initial minimization to not have a great impact on the final automata in our experiments. In the 31 experimental setups, a portion of which are shown in Table 3.2, only 4 yielded a different automaton depending on the initial minimization, and these differences were never more than 2 rules or states off. We thus indicate in Table 3.2, for each experiment, the number of states and rules initially, after one iteration of forward minimization, after one iteration of backward minimization, and after convergence, which was generally achieved after the sequence (backward, forward), equivalent to the sequence (forward, backward, forward). The differences between the two sequences, if any, were inconsequential.

Chapter 4

Machine Translation Model Contributions

> Translation is the other side of a tapestry
>
> Cervantes (attributed)

In this chapter we describe two instances of syntactic machine translation models that are expressed in terms of tree automata and tree transducers and use off-the-shelf generic algorithms to perform experiments that involve these models.

4.1 A tree-to-string transducer model for Japanese-to-English translation

Yamada and Knight describe a model of syntactic word-to-word translation from English trees to Japanese strings in (2001). They detail a method of using EM to learn word-to-word alignment from this model given a parallel corpus of English tree/Japanese string data, an example of which is in Figure 4.1. The translation model is depicted in Figure 4.2. According to the model described in (Yamada and Knight, 2001), English trees are transformed into Japanese strings via the following generative process that begins at the root of the English tree:

1. The node's children are re-ordered, that is, they are permuted probabilistically. For example, if there are three children, then there are six possible permutations whose probabilities add up to 1.
2. A decision is made about inserting a Japanese function word. This is a three-way decision at each node—insert to the left, insert to the right, or do not insert—and it depends on the labels of the node and its parent.
3. If the node's children are not leaves, recursively process them from step 1. Otherwise, English leaf words are translated probabilistically into Japanese, independent of context.

This model was encoded as a custom piece of software in (Yamada and Knight, 2001) and an EM algorithm specific to the task was used to learn weights for the model parameters, a sample of which are presented in Figure 4.3, given a training corpus. Naturally, the code written for this model was highly customized and thus inappropriate for any other EM learning tasks.

An approximate form of the (Yamada and Knight, 2001) story cast as a tree-to-string transducer was presented in (Graehl and Knight, 2004) alongside a generic EM algorithm for tree-transducers. A transduction begins at state q. From there a decision is made to insert a word, represented by state i, to either the left or the right of the current node, or to not insert a word. This is expressed as follows:

```
---------------------------------------------------------------
ENGLISH:    (VB (NN hypocrisy)
                (VB is)
                (JJ (JJ abhorrent)
                    (TO (TO to) (PRP them))))
JAPANESE:   kare ha gizen ga daikirai da
---------------------------------------------------------------
ENGLISH:    (VB (PRP he)
                (VB has)
                (NN (JJ unusual) (NN ability))
                (IN (IN in) (NN english)))
JAPANESE:   kare ha eigo ni zubanuke-ta sainou wo mot-te iru
---------------------------------------------------------------
ENGLISH:    (VB (PRP he)
                (VB was)
                (JJ (JJ ablaze)
                    (IN (IN with) (NN anger))))
JAPANESE:   kare ha mak-ka ni nat-te okot-te i-ta
---------------------------------------------------------------
ENGLISH:    (VB (PRP i)
                (VB abominate)
                (NN snakes))
JAPANESE:   hebi ga daikirai da
---------------------------------------------------------------

    etc.
```

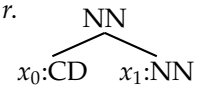
Figure 4.1: A portion of a bilingual tree/string training corpus.

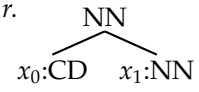
- $q.x_0 \rightarrow i.x_0\ r.x_0$
- $q.x_0 \rightarrow r.x_0\ i.x_0$
- $q.x_0 \rightarrow r.x_0$

From state i a Japanese word is generated regardless of the input, which can be a tree of arbitrary size:

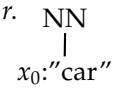
- $i.x_0 \rightarrow$ "de"
- $i.x_0 \rightarrow$ "kuruma"

From the reordering state r every permutation of every parent/child sequence is represented and the story continues from the beginning in state q:

- $r.$  $\rightarrow q.x_0\ q.x_1$

- $r.$  $\rightarrow q.x_1\ q.x_0$

At the English tree's pre-terminal we instead transition to state t to avoid any more insertion or reordering and prepare for translation. From state t we translate into some Japanese word or, alternately, no word:

- $r.$  $\rightarrow t.x_0$

- $t.$"car" $\rightarrow$ "kuruma"

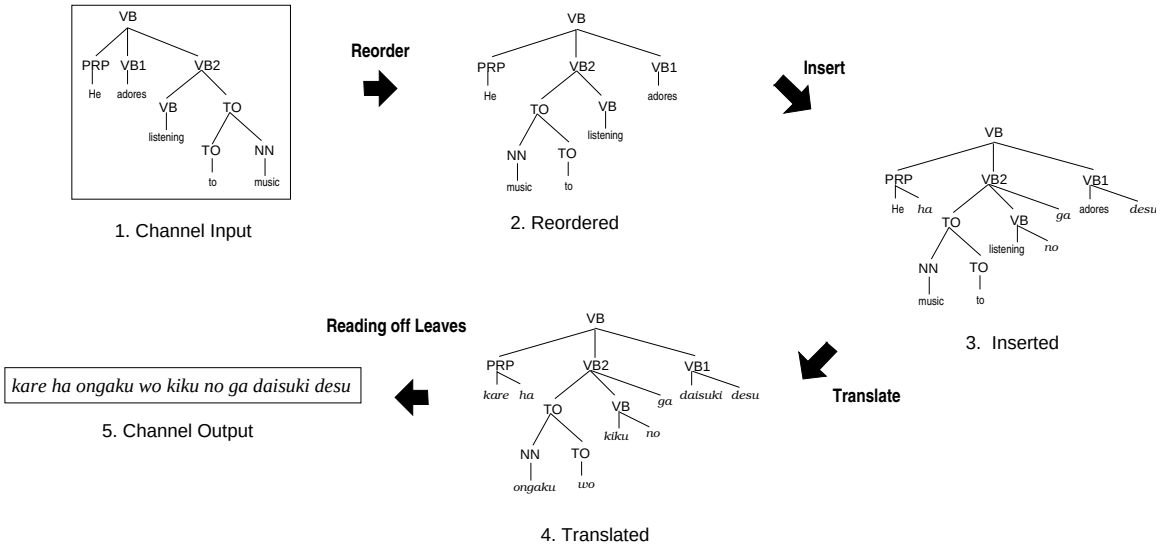
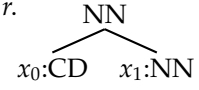
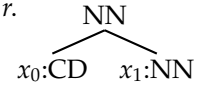
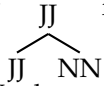
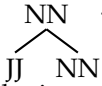


Figure 4.2: The translation model of Yamada & Knight, 2001.

- $t.\text{"car"} \rightarrow \epsilon$

Note the differences from the original model. Here we reorder conditioned on the parent; previously reordering was only done conditioned on the child sequence. Here, we decide whether to insert without context; previously insertion was conditioned on the previous and current symbols. Using these approximations results are rather dissimilar from the original model; the alignments learned by the approximation model match those learned by the original model in 53% of sentences. However, it is a fairly simple matter to correct these approximations, and even add more refinements to the model. In (Knight, Graehl, and May, 2008) we introduce a tree-to-string transducer model that exactly matches the model of (Yamada and Knight, 2001) and additionally includes an additional constraint present in the original C code but not described in the paper. To condition insertion on parent symbols we expand state q such that there is a separate state for each symbol. This affects both the insertion rules and the reordering rules:

- $q_{TOP}.x_0\text{:VB} \rightarrow i.x_0\ r.x_0$
- $q_{TOP}.x_0\text{:VB} \rightarrow r.x_0\ i.x_0$
- $q_{TOP}.x_0\text{:VB} \rightarrow r.x_0$
- $q_{VB}.x_0\text{:NN} \rightarrow i.x_0\ r.x_0$
- $q_{VB}.x_0\text{:NN} \rightarrow r.x_0\ i.x_0$
- $q_{VB}.x_0\text{:NN} \rightarrow r.x_0$
- $r.$  $\rightarrow q_{NN}.x_0\ q_{NN}.x_1$
- $r.$  $\rightarrow q_{NN}.x_1\ q_{NN}.x_0$

The reordering rules are still overspecified relative to the original model, which would not distinguish between the probability of reordering the children of  from  . Thus far we have estimated a separate parameter for each rule in the transducer. We change this by introducing rule *tying*, a mechanism

original order	reordering	P(reorder)
PRP VB1 VB2	PRP VB1 VB2	0.074
	PRP VB2 VB1	0.723
	VB1 PRP VB2	0.061
	VB1 VB2 PRP	0.037
	VB2 PRP VB1	0.083
	VB2 VB1 PRP	0.021
VB TO	VB TO	0.251
	TO VB	0.749
TO NN	TO NN	0.107
	NN TO	0.893
⋮	⋮	⋮

r−table

parent	TOP	VB	VB	VB	TO	TO	...
node	VB	VB	PRP	TO	TO	NN	...
P(None)	0.735	0.687	0.344	0.709	0.900	0.800	...
P(Left)	0.004	0.061	0.004	0.030	0.003	0.096	...
P(Right)	0.260	0.252	0.652	0.261	0.007	0.104	...

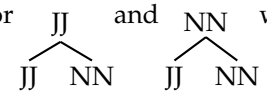
n−table

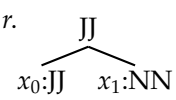
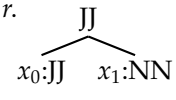
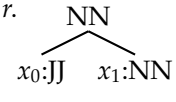
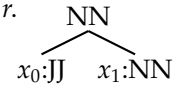
w	P(ins−w)
ha	0.219
ta	0.131
wo	0.099
no	0.094
ni	0.080
te	0.078
ga	0.062
⋮	⋮
desu	0.0007
⋮	⋮

E	adores		he		i		listening		music		to		···
J	*daisuki*	1.000	*kare*	0.952	*NULL*	0.471	*kiku*	0.333	*ongaku*	0.900	*ni*	0.216	···
			NULL	0.016	*watasi*	0.111	*kii*	0.333	*naru*	0.100	*NULL*	0.204	
			nani	0.005	*kare*	0.055	*mi*	0.333			*to*	0.133	
			da	0.003	*shi*	0.021					*no*	0.046	
			shi	0.003	*nani*	0.020					*wo*	0.038	
			⋮	⋮	⋮	⋮					⋮	⋮	

t−table

Figure 4.3: The parameter tables of Yamada & Knight, 2001.

for constraining more than one rule to be represented by the same parameter in training. By designating a set of transducer rules as tied we indicate that a single count collection and parameter estimation is performed for the entire set during training. We denote tied rules by marking each rule in the same tied class with the symbol @ and a common integer. Thus reordering rules for  and would appear as:

- *r.*  $\rightarrow q_{JJ}.x_0\ q_{JJ}.x_1$ @ 1
- *r.*  $\rightarrow q_{JJ}.x_1\ q_{JJ}.x_0$ @ 2
- *r.*  $\rightarrow q_{NN}.x_0\ q_{NN}.x_1$ @ 1
- *r.*  $\rightarrow q_{NN}.x_1\ q_{NN}.x_0$ @ 2

Unreported in (Yamada and Knight, 2001), the original code contained a constraint that specifically bars an unaligned foreign word insertion immediately prior to an ϵ English word translation. We incorporate this change to our model by simply modifying our transducer, rather than by changing our programming code, by introducing an additional state s, denoting a translation taking place immediately after an unaligned foreign function word insertion. For every rule that inserts a foreign function word we add an additional rule denoting an insertion immediately before a translation, and tie these rules together:

- $q_{TOP}.x_0$:VB $\rightarrow i.x_0\ r.x_0$ @ 23
- $q_{TOP}.x_0$:VB $\rightarrow i.x_0\ s.x_0$ @ 23
- $q_{TOP}.x_0$:VB $\rightarrow r.x_0\ i.x_0$ @ 24

model	states	initial rules	rules after training	training time (hours)	% link match	% sent. match
simple	4	98,033	12,413	16.95	87.42	52.66
exact	28	98,513	12,689	17.42	96.58	81.46
perfect	29	186,649	24,492	53.19	99.85	99.47

Table 4.1: A comparison of the three transducer models used to simulate the model of (Yamada and Knight, 2001). The "simple" model is the four-state approximation described in (Graehl and Knight, 2004). The "exact" model matches the description in (Yamada and Knight, 2001). The "perfect" model includes a derivation restriction not described in (Yamada and Knight, 2001) but incorporated in the actual code.

- $q_{TOP}.x_0$:VB $\rightarrow s.x_0\ i.x_0$ @ 24

To allow subsequent translation, "transition" rules for state s analogous to the transition rules described above must also be added. And, for each non-ϵ translation rule, we add an identical translation rule starting with s instead of t, and tie these rules:

- $s.$ NN(|x_0:"car") $\rightarrow s.x_0$
- t."car" $\rightarrow$ "kuruma" @ 14
- s."car" $\rightarrow$ "kuruma" @ 14
- t."car" $\rightarrow \epsilon$

We trained this model in Tiburon, which contains an implementation of the tree transducer training algorithm described in (Graehl and Knight, 2004) and (Yamada and Knight, 2001). The alignments learned by this model matched those learned by the model of (Yamada and Knight, 2001) in 99.47% instances and all disagreements were due to exact ties between two parameters. In fact, none of the parameter values learned by our reimplementation differed from the original parameter values by more than 0.00066. Table 4.1 contains complete results.

4.2 A new transducer model for syntactic re-alignment

The GHKM model of syntactic translation (Galley et al., 2004; Galley et al., 2006) is an extended tree-to-string transducer system. Rules such as those in Figure 4.4 accomplish phrase translation, reordering, and word insertion and deleting. The rules are single-state top-down rules with single symbol lookahead at their variable leaves, isomorphic to rules without lookahead but with a state for each looked-ahead type. Each rule has some probability, and the probability of all rules with the same left side root symbol sum to 1. A generative story for machine translation that computes $p(etree, f)$ given a foreign sentence f and a parsed target translation *etree* rooted at symbol v is:

1. Choose a rule r to replace v, with probability $p_{rule}(r|v)$.
2. For each variable with syntactic type v_i in the partially completed (tree, string) pair, continue to choose rules r_i with probability $p_{rule}(r_i|v_i)$ to replace these variables until there are no variables remaining.

As described in (Galley et al., 2004) we obtain transducer rules from parallel English-foreign sentences, heuristic English parses, and word-to-word alignments. The last of these is generally realized by unsupervised (Och and Ney, 2000) or semi-supervised (Fraser and Marcu, 2006; Fraser and Marcu, 2007) models that are syntax-unaware. However, the generative story described above can be thought of as an alignment model as well, as each rule carries implicit partial alignment between all lexical elements. The set of rules in a single derivation implies a complete word alignment for the derived sentence. Given a set of English

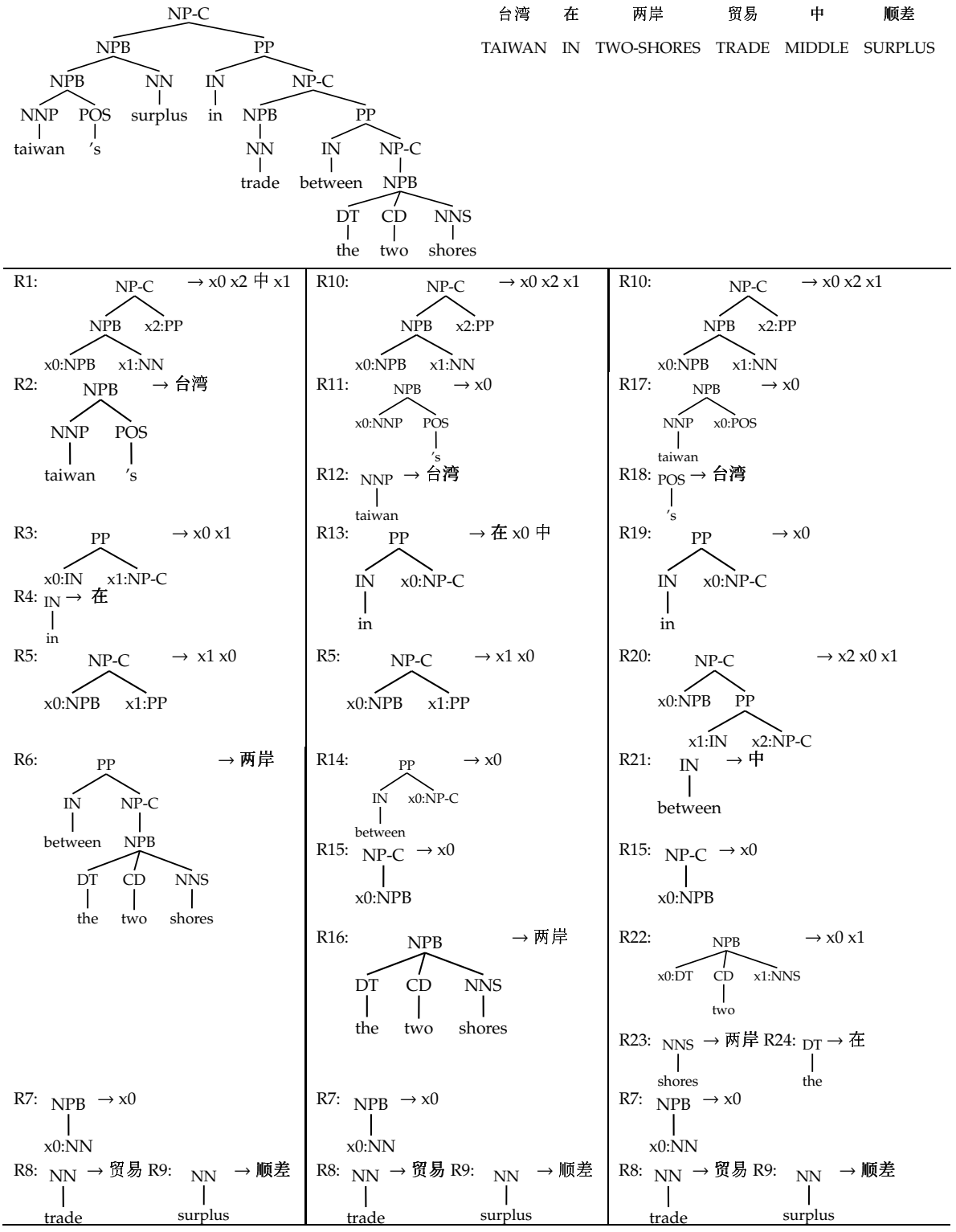


Figure 4.4: A (English tree, Chinese string) pair and three different sets of multilevel tree-to-string rules that can explain it.

GIZA		-			
E	C				
9,864,294	7,520,779	baseline	19,138,252	39.08	37.77
		re-alignment	26,053,341	**39.76**	**38.69**
221,835,870	203,181,379	baseline	23,386,535	39.51	38.93
		re-alignment	33,374,646	**40.17**	**39.96**

(a) Chinese re-alignment corpus has 9,864,294 English and 7,520,779 Chinese words

GIZA		-			
E	A				
4,067,454	3,147,420	baseline	2,333,839	**47.92**	47.33
		re-alignment	2,474,737	47.87	**47.89**
168,255,347	147,165,003	baseline	3,245,499	49.72	49.60
		re-alignment	3,600,915	49.73	**49.99**

(b) Arabic re-alignment corpus has 4,067,454 English and 3,147,420 Arabic words

Table 4.2: Machine translation experimental results from (May and Knight, 2007) evaluated with case-insensitive BLEU4.

tree/foreign string pairs and a large set of transducer rules such as those in Figure 4.4 an unsupervised training algorithm can learn the most useful rules across the corpus and thus learn syntax-informed word alignments.

In (May and Knight, 2007) we evaluate the impact of this alignment method on Arabic-English and Chinese-English translation experiments using the training algorithm for tree transducers of (Knight, Graehl, and May, 2008). We found that, while we were able to get modest improvements using the generative story above, results improved by incorporating a parameter based on the size of the rules being used, where size is the number of non-leaf nodes on the left side of the rule. Our revised story is:

1. Choose a size s with cost $c_{size}(s)^{s-1}$
2. Choose a rule r of size s to replace v, with probability $p_{rule}(r|v,s)$.
3. For each variable with syntactic type v_i in the partially completed (tree, string) pair, continue to choose sizes s_i followed by rules r_i with probability $p_{rule}(r_i|v_i,s_i)$ to replace these variables until there are no variables remaining.

This modification effectively ensures that competing derivations of the same training pair have the same number of parameters. As an example, consider the pair (NPB / NNP POS / taiwan 's , 台湾). Given the set of rules from Figure 4.4 the three possible derivations are R2, R11-R12, and R17-R18. If we consider each instance of the cost $c_{size}(s)$ as a separate parameter, the same derivations according to the modified story are $c_{size}(3)$-$c_{size}(3)$-R2, $c_{size}(2)$-R11-R12, and $c_{size}(2)$-R17-R18. This modification helps alleviate EM's tendency to overfit to larger rules in shorter derivations. It is accomplished by a simple modification to the derivation RTG and does not require any change to the learning model.

Using this generative story we were able to obtain re-alignments that, when used in a syntactic machine translation system (Galley et al., 2004; Galley et al., 2006; DeNeefe et al., 2007) lead to consistent BLEU improvements in Chinese-English and Arabic-English translation experiments. We obtain these improvements even when we improve the quality of the bootstrapped alignments by drastically increasing the amount of data used in GIZA. Table 4.2 contains complete results.

Chapter 5

Status of Current Work and Anticipation of Future Work

> Past performance is no guarantee of future results
>
> Standard disclaimer

Most of the work described in Chapters 3 and 4 has been presented to the community. Weighted determinization of tree automata was presented at NAACL (May and Knight, 2006a). Bisimulation minimization results were presented at DLT (Högberg, Maletti, and May, 2007c) and CIAA (Högberg, Maletti, and May, 2007a), as well as in two technical reports (Högberg, Maletti, and May, 2007b; Högberg, Maletti, and May, 2007d), and a journal version has also been prepared (Högberg, Maletti, and May, 2008). The first alpha version of Tiburon was released in 2006 concurrent with its descriptive paper at CIAA (May and Knight, 2006b). Since that time many improvements have been added and subsequent versions released. The software has been used in graduate-level natural language processing classes and we know of at least one thesis that used Tiburon to evaluate experiments. Tiburon was also used to replicate the work presented in (Yamada and Knight, 2001) as discussed in Chapter 4; this replication is detailed in the CL journal (Knight, Graehl, and May, 2008). Finally, the re-alignment model was proposed and evaluate in work published at EMNLP (May and Knight, 2007).

However, there are some important holes in this research effort that I hope to fill before presenting the completed thesis. This chapter details the work to be completed and an approximate timetable in which I will complete the work.

5.1 Algorithmic improvements

5.1.1 Determinization

A principal deficit in the algorithm presented in (May and Knight, 2006a) is its unsuitability for some automata with infinite languages, caused by cycles in an automaton's hypergraph. (Mohri, 1997) noted that only cyclic FSAs with the "twins" property are able to be determinized. Without a twins property for weighted tree automata we were unable to determine the subset of determinizable cyclic weighted tree automata. Subsequently (Allauzen and Mohri, 2004) described an algorithm based on the twins property that allows weighted cyclic FSAs without the twins property to be determinized. As such an algorithm is quite useful for determinization of weighted tree automata, I intend to extend both the twins property and pre-determinization algorithm to the tree case.

5.1.2 Minimization

It is important to minimize automata after determinization to lessen the effects of state explosion caused by determinization. An algorithm to minimize deterministic weighted tree automata is presented in (Maletti, 2008) but no implementation of this algorithm is known to exist, so I will implement it in Tiburon. The algorithm in (Maletti, 2008) does not use a local heuristic such as the "pushing" described in (Mohri, 1997). Such a local heuristic would lead to faster minimization, so I will investigate whether it is possible to obtain a pushing minimization algorithm.

5.1.3 Composition and related algorithms

It has been well established that top-down tree transducers may be composed to the left of top-down, non-deleting, linear tree transducers (Gécseg and Steinby, 1984). However, as I am unaware of a specific implementable algorithm for performing this composition, I have devised one. I will present the algorithm and prove its correctness. I have also engaged in some recent work with collaborators on the ability to detect whether two particular instances of a class of transducers generally not closed under composition are in fact composable, such as extended linear non-deleting top-down tree-to-tree transducers. If we are able to detect such instances of specific composability, I will add the relevant algorithms to Tiburon. This also may raise other issues. Transducers with rules that change state without consuming input or producing output, so-called ϵ-rules, are not in general closed under composition but they can be quite useful, and it is likely that many instances are in fact closed under composition. However, as is shown for the string case in (Pereira and Riley, 1997), when compositions of this sort are done in a naïve manner with weighted transducers, incorrect composition results can occur unless corrections are made. I intend to apply this type of correction to tree transducer composition when ϵ-rules are used.

Obtaining the domain projection of a general top-down tree transducer as a finite-state tree automaton is a desirable property, as we may wish to know the input language of some transformation or obtain the result of a backward application via composition. As was the case in the previous section, it is known that the domain of any top-down tree transducer is a regular tree language and thus representable by a tree automaton (Gécseg and Steinby, 1984) but an explicit algorithm for obtaining the automaton is not, to our knowledge, available. I will present such an algorithm and prove its correctness.

5.2 Software improvements

Tiburon has a useful set of algorithms already implemented but some already known algorithms have not yet been implemented. These include backward application of strings or string automata through tree-to-string transducers, *n*-best generation of *pairs* from a tree transducer, and EM training of string automata. Table 5.1 identifies currently implemented, unimplemented, and undeveloped algorithms for Tiburon. Not included in this table are algorithms for string automata and transducers, whose algorithms are superseded by the algorithms of Tiburon, and thus should be representable and handleable.

A principal deficiency in Tiburon is its relative slowness, especially in comparison to mature string automata toolkits such as OpenFST and Carmel. I have until now neglected efficiency improvements in favor of feature improvements, but this has led to less utility in Tiburon than is desired. For example, the re-alignment results of (May and Knight, 2007), while conceivably obtainable by using Tiburon, were in fact obtained with custom software that builds derivation forests in a bottom-up manner (as opposed to via the top-down derivation forest construction algorithm specified in (Knight, Graehl, and May, 2008)). The custom software was also highly optimized to be memory and time efficient. I will conduct a deep investigation into the secrets behind the speed of OpenFST, including taking advantage of efficient data structures and implementing different algorithms, such as bottom-up derivation forest construction and lazy composition where appropriate in an attempt to bring Tiburon to a level of engineering comparable to that used for the experiments of (May and Knight, 2007).

Algorithm	CFG	RTG	Tree-Tree	Tree-String
domain projection	N/A	N/A	Yes	Yes
range projection	N/A	N/A	Soon	Soon
EM training	Yes	Yes	Yes	Yes
k-best	Yes	Yes	Soon	Soon
stochastic generation	Yes	Yes	Soon	Soon
intersect	Imp	Yes	Imp	Imp
Forward application of tree to...	N/A	N/A	Yes	Yes
Forward application of RTG to...	N/A	N/A	Yes	Yes
Backward application of tree to...	N/A	N/A	Yes	N/A
Backward application of string to...	N/A	N/A	N/A	Soon
Backward application of CFG to...	N/A	N/A	N/A	Soon
Composition of unextended linear nondeleting...	N/A	N/A	Yes	N/A
Composition of extended linear nondeleting...	N/A	N/A	Rsrch	N/A
Bisim. minimization	Soon	Soon	Rsrch	Rsrch
True minimization	Soon	Soon	Rsrch	Rsrch
Determinization	Rsrch	Yes	Rsrch	Rsrch
coerce string to...	Soon	N/A	N/A	N/A
coerce tree to...	Yes	Yes	Yes	Yes
coerce CFG to...	N/A	Yes	Yes	Yes
coerce RTG to...	Yes	N/A	Yes	Yes
coerce tree-to-tree to linear...	N/A	N/A	Rsrch	N/A
coerce tree-to-string to linear...	N/A	N/A	N/A	Rsrch

Table 5.1: Status of features in Tiburon. Yes = the feature is present in current version 0.5.0. Soon = the feature has yet to be implemented, but a clear path to its implementation is known. Rsrch = more research must be done before an algorithm can be determined and implemented. Imp = the feature is known to be an undecidable algorithm or impossible to construct (this does not preclude limited local tests). N/A = the algorithm is not applicable to this machine

5.3 Experiment improvements

The results on re-alignment described in (May and Knight, 2007) were presented over relatively small training corpora compared to the corpora used for high-quality GIZA bootstraps. We believe we can get even better performance from re-alignment by increasing the size of the training corpus but have not been able to satisfactorily engineer our code to handle data of this size. We intend to improve our system to handle two orders of magnitude more data than reported in (May and Knight, 2007).

The empirical results of bisimulation minimization presented in (Högberg, Maletti, and May, 2007a; Högberg, Maletti, and May, 2007c; Högberg, Maletti, and May, 2007b; Högberg, Maletti, and May, 2007d) showed that bisimulation had an appreciable effect on automata that recognized real trees, but the justification for the particular automata as they could be used in a practical NLP experiment is weak, given that there is currently little call for tree language models, and less call for nondeterministic automata when quick membership checking is desired. However, it may be possible to improve the weighted determinization results of (May and Knight, 2006a) as those results were based on determinization completing within a preset amount of processing time. A bisimulation-minimized input could be determinized more quickly and thus complete in the allotted time, leading to more reorganization of *k*-best lists, and better performance. We will perform this experiment to help validate both algorithms.

5.4 Additional modeling work

In (Knight, Graehl, and May, 2008) we described how tree machine formalisms could be used for clean representations of the complicated syntactic model of (Yamada and Knight, 2001). If a tree transducer toolkit is to be useful it should be applicable to a wide variety of syntax models. We will investigate how

tree machines in particular and Tiburon in particular may be used for parsing models such as those in (Collins, 1997), (Charniak, 2001), and (Petrov and Klein, 2007).

5.5 Timeline to completion

In the early Summer of 2008 I have been engaged in proposal preparation activities. Through the summer I will begin on more advanced re-alignment experiments in preparation of a journal paper dealing with multiple transformations of training data for improved machine translation (joint work with Wei Wang, Kevin Knight, and Daniel Marcu). Also in the summer I will finalize a minor release of Tiburon, 0.5.0, which contains implementations of projection and composition, using the previously described new algorithms.

In the Fall of 2008 I will conduct an experiment on using bisimulation minimization before determinization to obtain faster determinization and thus acquire higher quality k-best output lists. I will also work on a major release of Tiburon, version 1.0, which will contain all currently known implementation, especially including backward application of strings and CFGs onto tree-to-string transducers, essentially the ability to parse. I will also make Tiburon backward compatible with classic string machine algorithms, and thus support string transducer and string automaton formats.

In the Spring of 2009 I will turn to more algorithmic matters. I will focus on determinization and minimization algorithm improvements. This will hopefully lead to improved algorithms added to Tiburon. I will also work on an investigation of parser models, expressing them in the formalisms described here to demonstrate the powers and limitations of tree transducers and automata.

Finally, in the Summer of 2009 I will work on engineering efforts to improve the performance of Tiburon. This includes an investigation into the codebase of OpenFST (Riley et al., 2007) and exploration of lazy algorithms and coordinated search in lieu of explicit composition. The end result will be an optimized Tiburon 2.0. I will then allot a month at the end of the summer to finish composing the final thesis (it is my hope I will be composing all along this process) and finally defend. This timeline is summarized in Table 5.2.

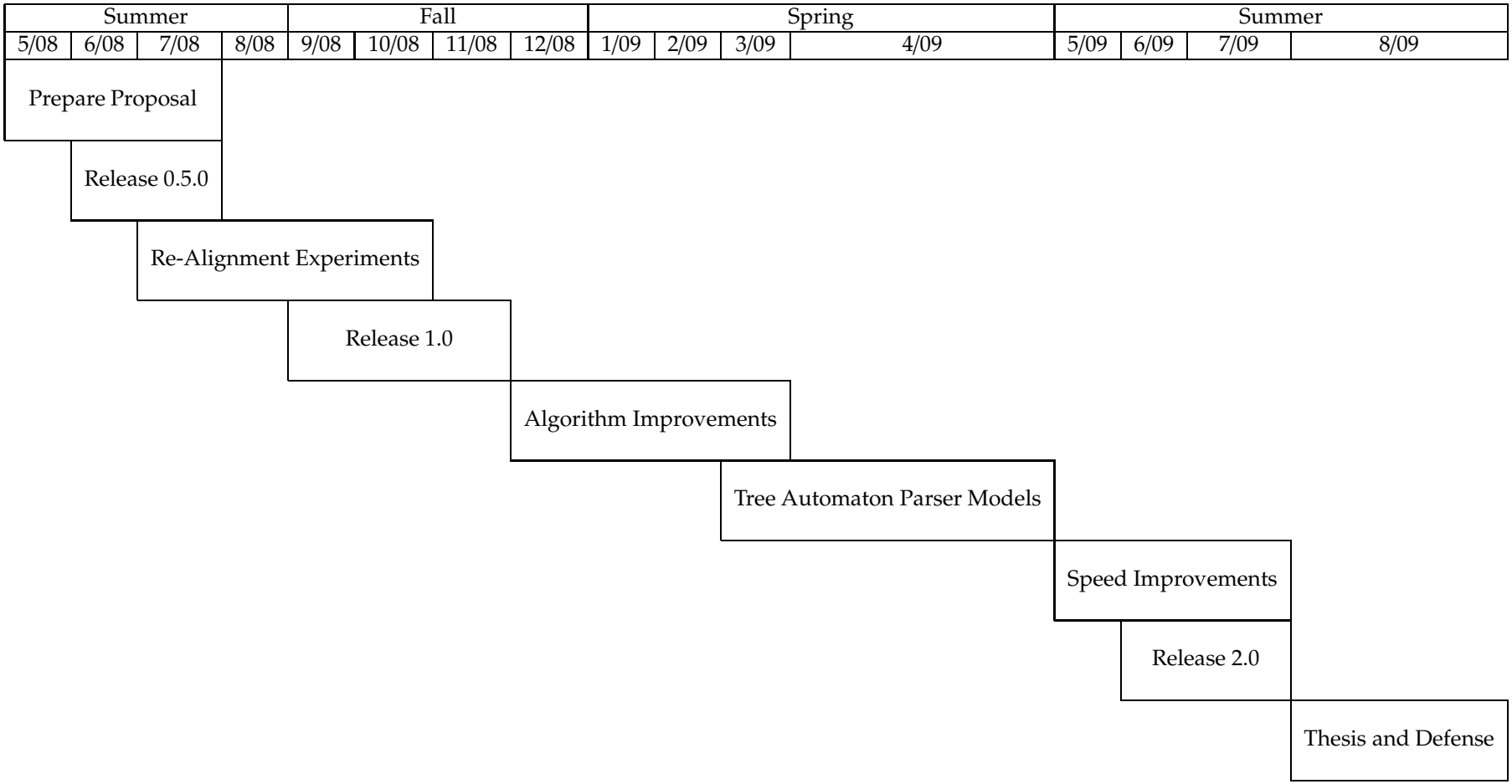


Table 5.2: Overview of proposed schedule

References

> Those who quote others have
> nothing to say for themselves.
>
> David Luméz

Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass.

Allauzen, Cyril and Mehryar Mohri. 2004. An optimal pre-determinization algorithm for weighted transducers. *Theoretical Computer Science*, 328(1–2):3–18.

Allauzen, Cyril and Mehryar Mohri. 2007. N-way composition of weighted finite-state transducers. Technical Report TR2007-902, Department of Computer Science, Courant Institute of Mathematical Sciences, New York University, August.

Arbib, M. A. and Y. Give'on. 1968. Algebra automata I: Parallel programming as a prolegomena to the categorical approach. *Information and Control*, 12(4):331–345, April.

Baker, Brenda S. 1979. Composition of top-down and bottom-up tree transductions. *Information and Control*, 41(2):186–213, May.

Bod, Rens. 2003. An efficient implementation of a new DOP model. In *Proc. EACL*, Budapest.

Borchardt, Björn and Heiko Vogler. 2003. Determinization of finite state weighted tree automata. *Journal of Automata, Languages and Combinatorics*, 8(3).

Borovansky, P., C. Kirchner, H. Kirchner, P.E. Moreau, and M. Vittek. 1996. Elan: A logical framework based on computational systems. In *Proceedings of the first international workshop on rewriting logic*.

Brainerd, Walter S. 1968. The minimalization of tree automata. *Information and Control*, 13(5):484–491, November.

Charniak, Eugene. 2001. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 116–123.

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

Chomsky, Noam. 1957. *Syntactic Structures*. Mouton.

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

Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *ACL Proceedings*.

Comon, H., M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 2007. Tree automata techniques and applications. release October, 12th 2007.

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

DeNeefe, Steve, Kevin Knight, Wei Wang, and Daniel Marcu. 2007. What can syntax-based MT learn from phrase-based MT? In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Conference on Computational Natural Language Learning*, Prague, Czech Republic, June 28–30.

Doner, John. 1970. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4(5):406–451, October.

Echihabi, Abdessamad and Daniel Marcu. 2003. A noisy-channel approach to question answering. In *ACL Proceedings*.

Eisner, Jason. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proc. ACL*, Philadelphia, July.

Eisner, Jason. 2003. Simpler and more general minimization for weighted finite-state automata. In *Proc. HLT-NAACL*, Edmonton, May.

Engelfriet, Joost, Zoltán Fülőp, and Heiko Vogler. 2001. Bottom-up and top-down tree series transformations. *Journal of Automata, Languages and Combinatorics*, 7(1):11–70, July.

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

Fraser, Alexander and Daniel Marcu. 2006. Semi-supervised training for statistical word alignment. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 769–776, Sydney, Australia, July 17–21.

Fraser, Alexander and Daniel Marcu. 2007. Getting the structure right for word alignment: LEAF. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 51–60, Prague, June.

Galley, Michel, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steven DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic models. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 961–968, Sydney, Australia, July 17–21.

Galley, Michel, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *HLT-NAACL Proceedings*.

Gécseg, Ferenc and Magnus Steinby. 1984. *Tree Automata*. Akadémiai Kiadó, Budapest.

Genet, T. and V. Viet Triem Tong. 2001. Reachability analysis of term rewriting systems with timbuk. In *LPAR Proceedings*.

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

Graehl, Jonathan and Kevin Knight. 2004. Training tree transducers. In *Proc. NAACL-HLT*.

Henriksen, J.G., J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm. 1995. Mona: Monadic second-order logic in practice. In *TACAS Proceedings*.

Högberg, Johanna, Andreas Maletti, and Jonathan May. 2007a. Backward and forward bisimulation minimisation of tree automata. In Jan Holub and Jan Žďárek, editors, *Proc. 12th Int. Conf. Implementation and Application of Automata*, volume 4783 of LNCS, pages 109–121, Prague. Springer.

Högberg, Johanna, Andreas Maletti, and Jonathan May. 2007b. Backward and forward bisimulation minimisation of tree automata. Technical Report ISI-TR-633, University of Southern California.

Högberg, Johanna, Andreas Maletti, and Jonathan May. 2007c. Bisimulation minimisation for weighted tree automata. In Tero Harju, Juhani Karhumäki, and Arto Lepistö, editors, *Proc. 11th Int. Conf. Developments in Language Theory*, volume 4588 of LNCS, pages 229–241, Taipei. Springer.

Högberg, Johanna, Andreas Maletti, and Jonathan May. 2007d. Bisimulation minimisation of weighted tree automata. Technical Report ISI-TR-634, University of Southern California.

Högberg, Johanna, Andreas Maletti, and Jonathan May. 2008. Backward and forward bisimulation minimisation of tree automata. *Theoretical Computer Science*. submitted.

Hopcroft, J. E. 1971. An $n \log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi, editor, *Theory of Machines and Computations*. Academic Press.

Huang, Liang and David Chiang. 2005. Better k-best parsing. In *Proc. IWPT*, Vancouver.

Joshi, Aravind K. and Phil Hopely. 1996. A parser from antiquity. *Natural Language Engineering*, 2(4):291–294.

Kaplan, Ronald and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*.

Karttunen, L., J. P. Chanod, G. Grefenstette, and A. Schiller. 1997. Regular expressions for language engineering. *Natural Language Engineering*, 2(4):1–24.

Karttunen, Lauri, Tamás Gaál, and André Kempe. 1997. Xerox finite-state tool. Technical report, Xerox Research Centre Europe.

Knight, Kevin and Yaser Al-Onaizan. 1998. Translation with finite-state devices. In *AMTA Proceedings*.

Knight, Kevin, Jonathan Graehl, and Jonathan May. 2008. Training tree transducers. *Computational Linguistics*, 34(3). To Appear.

Knight, Kevin and Daniel Marcu. 2002. Summarization beyond sentence extraction: A probabilistic approach. *Artificial Intelligence*, 139.

Kolak, Okan, William Byrne, and Philip Resnik. 2003. 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*, pages 55–62, Edmonton, Canada, May-June. Association for Computational Linguistics.

Koskenniemi, Kimmo. 1983. Two-level morphology: A general computational model for word-form recognition and production. Publication 11, University of Helsinki, Department of General Linguistics, Helsinki.

Kumar, Shankar and William Byrne. 2003. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *HLT-NAACL Proceedings*.

Maletti, Andreas. 2008. Minimizing deterministic weighted tree automata. In *Proc. LATA*, Tarragona, Spain, March.

Maletti, Andreas, Jonathan Graehl, Mark Hopkins, and Kevin Knight. 2008. On extended tree transducers. Under Review.

Mathias, Lambert and William Byrne. 2006. Statistical phrase-based speech translation. In *IEEE Conference on Acoustics, Speech and Signal Processing*.

May, Jonathan and Kevin Knight. 2006a. 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. Association for Computational Linguistics.

May, Jonathan and Kevin Knight. 2006b. 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. Springer.

May, Jonathan and Kevin Knight. 2007. Syntactic re-alignment models for machine translation. In Jason Eisner and Taku Kudo, editors, *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 360–368, Prague, Czech Republic, June 28 – June 30. Association for Computational Linguistics.

Meyer, A. R. and L. J. Stockmeyer. 1972. The equivalence problem for regular expressions with squaring requires exponential space. In *Proc. 13th Annual Symp. Foundations of Computer Science*, pages 125–129. IEEE Computer Society.

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

Mohri, Mehryar. 2002. Generic ϵ–removal and input ϵ–normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143.

Mohri, Mehryar, Fernando C. N. Pereira, and Michael Riley. 2000. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231:17–32, January.

Mohri, Mehryar and Michael Riley. 2002. An efficient algorithm for the n-best strings problem. In *Proc. ICSLP*.

Och, Franz and Hermann Ney. 2000. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 440–447, Hong Kong, October 1–8.

Pereira, Fernando and Michael Riley. 1997. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*. MIT Press, Cambridge, MA, chapter 15.

Pereira, Fernando, Michael Riley, and Richard Sproat. 1994. Weighted rational transductions and their application to human language processing. In *Human Language Technology*, pages 262–267, Plainsboro, NJ, March. Morgan Kaufmann Publishers, Inc.

Petrov, Slav and Dan Klein. 2007. Learning and inference for hierarchically split PCFGs. In *AAAI 2007 (Nectar Track)*.

Rabin, M. O. and D. Scott. 1959. Finite automata and their decision properties. *IBM Journal of Research and Development*, 3(2):114–125, April.

Riley, M., J. Schalkwyk, W. Skui, C. Allauzen, and M. Mohri. 2007. OpenFst library. http://www.openfst.org.

Rounds, William C. 1970. Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287.

Schützenberger, M. P. 1961. On the definition of a family of automata. *Information and Control*, 4:245–270.

Shannon, Claude. 1948. A mathematical theory of communication. *Bell System Technical Journal*, 27.

Sproat, Richard, William Gales, Chilin Shih, and Nancy Chang. 1996. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3), September.

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

Yamada, Kenji and Kevin Knight. 2001. A syntax-based statistical translation model. In *ACL Proceedings*, pages 523–530.

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